

# PROCEEDINGS



**THE INFORMATION CENTRE  
AND CHANGING TECHNOLOGIES**



**1984 APL USERS MEETING**

**OCT. 15-17 TORONTO, CANADA**



I.P. Sharp Associates



# **The Information Centre and Changing Technologies**

**Proceedings of the 1984 APL Users Meeting**

**I.P. Sharp Associates Limited  
Toronto, Ontario  
October, 1984**

Printed in Canada

October 1984

Publication code: 0382 8410 E1

ISBN 0 86493 098 4



## CONTENTS

Introducing a Global Information Centre Lib Gibson	1
Implementing a Global Information Centre Eric B. Iverson	15
Evaluating and Implementing User Productivity Tools Robert Metzger	27
Managing the Information Revolution David Keith, James L. Schmit, Keith W. Iverson, Gösta Olvai, and Eric B. Herr	77
Managing and Implementing International Systems: The Human Factor W. Lee Bettes, Sheila E. Goldman, and Paul M. Blake	119
Some Management Tools for the Information Centre William Apsit	131
LOGOS: An APL Programming Environment David B. Allen, Mark R. Dempsey, Leslie H. Goldsmith, and Kevin L. Harrell	173
The SHARP APL Environment Richard Lathwell	201
Exploiting Networks Joey Tuttle	207



# **INTRODUCING A GLOBAL INFORMATION CENTRE**

Lib Gibson, Director, Application Software Development, I. P. Sharp Associates

## **Abstract**

This keynote address identifies four major challenges to business today: the transition from an industrial to an information society; the reduction in the time an organization has to adapt to change; the emergence of a global economy; and the need for increased people productivity. Technology can help address these problems through provision of data, communications, and software. It is the true integration of these elements that is a Global Information Centre.



## **BUSINESS CHALLENGES OF THE 80's**

We at I. P. Sharp Associates are in the business of using technology to solve business problems. As we deal with organizations, we become familiar with their problems, discuss their needs and expectations, and participate in their efforts to use technology to address those concerns. As we work with organizations around the world, we have seen the emergence of problems which are common to them all. Some companies regard these problems as threats and face them with trepidation; others view them as challenges and greet them as exciting opportunities. One can't help but expect the latter to be the ones who will survive and prosper in these exciting and turbulent times.

What then are these challenges which face today's organizations?

### **Moving From An Industrial Society To An Information Society**

The fundamental challenge is to make the transition from an industrial society to an information society. The companies which flourished in the industrial age were those which harnessed the new industrial machines to produce more products, better products, and cheaper products. They built a better mousetrap and the world beat a path to their doorway. But today, simply building a better mousetrap is not enough. You must know how to market that mousetrap effectively, hedge on cheese futures, borrow capital at optimum rates, and, finally, plan future products which will provide revenue when the popularity of mousetraps has led to the total extermination of mice. Even if the passion of your existence is the manufacture of mousetraps, you ignore these other factors at your peril.

All these other factors affecting your business are contingent on information: market research results, interest rate trends, sales history and forecasts, commodity market future prices, balance sheets, economic indicators and much else. Considerable skill is needed for the assimilation and analysis of this wide range of information and a considerable proportion of time must be spent on it. For success, this effort must be effective.

### **Reduced Reaction Time**

To complicate matters, the pace of change is accelerating, leaving less and less time to absorb all this information and react to it. As early as the sixties, David Bell pointed out that the average time span between the initial discovery of a new technological innovation and the recognition of its commercial potential had shrunk from three decades to less than one. And that trend is continuing. This is a second major concern of organizations: to be able to react swiftly to changes in the environment. Almost two hundred years ago, the Rothschilds took spectacular advantage of early knowledge of the outcome of the battle of Waterloo; carrier pigeons gave them a flying head start of several hours, which paid off handsomely in the financial markets. Adventurous speculators today sniff out arbitrage

opportunities that may last only a few minutes or even seconds. Buggy manufacturers had decades to adapt to the arrival of the automobile; today's software manufacturers have at most months to adapt to startling advances in hardware.

### **The Global Economy**

The third challenge is to recognize the existence of the global village. No longer is it enough to limit one's vision to analysis of one's local or even national economy. The marketplace for goods and capital is now worldwide. There is tremendous interdependence within the world: economic and political actions can have serious ramifications around the world. Currency fluctuations in a country on the other side of the world can make that nation's products cheaper in its home market and seriously damage the competitiveness of a product abroad; on the other hand, a fluctuation in the other direction may expand the market for a product so swiftly that supply may not keep up with demand. The politico-economic actions of oil producing nations may severely impact supply of raw materials, or dramatically affect overall costs. Every business is affected and has to understand the global situation.

For a multinational company, the situation is even more complicated. How can resources be deployed internationally so as to maximize effectiveness? How can a company remain decentralized, affording local entities freedom to succeed in local conditions, and yet exercise sufficient central control to act in a coordinated and purposeful way? This is a difficult dilemma to resolve.

### **Raising Productivity**

But, finally, when all is said and done, a company is its people. And the fourth challenge for a company today is to bring out the best in its people, and to raise their productivity. In the industrial period, productivity took dramatic leaps through such innovations as assembly lines. Efficient they were; enjoyable or fulfilling they usually were not. We are more fortunate in the information age. Gains in productivity will be made through efficiency of information processing; and we in the information processing industry know how much fun that will be! A whole generation of computer-literate kids agrees. Computer technology promises the best opportunity for major productivity gains. It is instrumental in making people more effective in coping with the vast amount of information available, assimilating and analysing it, and acting wisely based on it.

### **Changing Technologies**

What does technology offer to address these challenges? It offers data, communications, and software. These are the changing technologies which form the theme of this conference. Let's look at each resource in detail.

**Data**

There are several types of data:

internal or external  
and  
numeric or textual.

Internal, or corporate data as it is more commonly known, is data pertaining to the organization itself. Much of this data is numeric, such as budgets, plans, sales history and projections, financial statistics, and production figures. The data covers a wide range of detail, from the overall balance sheet for the corporation, which might be worldwide in scope, down to data that may be relevant and useful to one department, or indeed only one person in the organization. Numeric data is most useful when there is history available from which one can discern trends and try to forecast the future.

Other corporate data is textual, such as customer lists, contracts, product descriptions, employee lists, and reports. Increasingly, electronic mail represents a further vital collection of textual data for any organization.

The management of this corporate data is a critical and burdensome task. Since the data is completely internal to the organization, it is necessary for the corporation to collect it, which is far easier to say than to do, especially when we talk of international companies. Then the information must be disseminated to those who need it, but withheld from those who shouldn't have access.

External, or public data is rather different. It might be numeric, such as stock market prices on the New York, London, and Hong Kong stock exchanges, airline traffic between Dallas and Chicago, gross national product's of OECD nations, crude oil production in Iran, the number of housing starts in Saskatoon, or money supply figures for Germany. As with internal data, history is desirable, so trends can be picked out.

External textual data takes many forms. Of course, the first thing that leaps to mind is the mass of data available from newspapers, magazines and journals, and the electronic bibliographic systems which help one to search and select from this vast amount of material. Systems range from Chemical Abstracts services, to legal and legislative data bases, to online newspapers. Electronic publishing of newsletters, where timeliness is of the essence, is a fast growing field. In this case, rather than the electronic version being a secondary copy of the printed publication, the electronic version is in fact the primary source.

Unlike internal data, the burden of data collection and data integrity for external data rests with some external data supplier. The difficulty for an organization is in finding it, and marrying it with internal data. There is a session later in this conference "Managing the Information Revolution" which deals further with these four types of data. Keith Iverson

from Morgan Stanley talks about internal numeric data and Gösta Olavi describes Skandinaviska Enskilda Banken's treatment of one particular kind of internal textual data, electronic mail. James Schmit of Compu Trac discusses provision of external numeric data while Eric Herr of McGraw-Hill refers to electronic publishing, or the province of external textual data.

If we think back to our original challenges, how can data help address them? Firstly, to meet the need for sound information, data integrity of course is paramount. The organization must strive to ensure that their internal data is accurate, and that everyone in the organization deals with that accurate data. There's no point in having correct data on a mainframe if managers in the field are basing decisions on their own out-of-date versions of the data. It is also necessary to critically evaluate external data to ensure it meets the same standard of integrity.

Secondly, the organization must make sure that data is timely. Data is a very perishable commodity. With the dizzying pace of business forcing quick decisions, there's no point in delivering the balance sheet information to the treasurer in two days, or even two hours, if he has to decide whether to roll over a financial deal **now**. The organization as a whole must be made aware of the importance of timeliness. This is often difficult since the 'providers' of the data may not be the ultimate 'consumers' of the data, and thus may be difficult to motivate. This is particularly true in the case of subsidiaries reporting to headquarters. Suggestions on how to deal with some of these problems are addressed in the paper on "Managing and Implementing International Systems". External sources of data must be subjected to the same close scrutiny.

Thirdly, the scope of the data must be broad enough to support a global outlook. If the company is international, data from the whole organization must be available as a coherent whole. And the corporate data must be complemented by external data of international scope.

Lastly, the data must be accessible, conveniently, for those who need it if they are to address the challenge of productivity. Users must have accessibility to common data. For the corporate organism to act in coordination, all its parts must be reacting to the same stimulus, that is, the same data. Consider the uncoordinated behaviour of a person trying to write while watching their writing reflected in a mirror. An organization must protect against similar discordant behaviour by ensuring that everyone receives the same data. The only way to be absolutely sure of this is to give everyone access to the same shared data source. They not only use the same data, but any revisions or corrections of inaccuracies are shared by all.

The facilitator of this accessibility is the electronic communications network, which brings data in reach of all users.



## **Communications**

This leads us directly to our second technological resource—communications, or, in other words, an electronic network. An overview of network technology is presented in the session “Exploiting Networks”. An electronic network can help with both the collection and the dissemination of data. It can provide access to large amounts of stored historical data in a form in which it can be selectively searched and used. Or, it can provide virtually instant transmission of a single critical number across the world. In different environments it may be involved in one-way communication for either collection or dissemination of data, or two-way interactive communications, supported by various software tools. A network can be used in the transmission of ideas, for this is exactly what is involved in electronic mail or conferencing. Electronic mail cuts across geographical and hierarchical roadblocks and allows people to exchange ideas and information freely and effectively.

All these attributes make communications an important element in addressing business challenges. Communications provide a solid, consistent information base for a company. Without a good communications system to tightly link a company, it is almost certain that different parts of the organization will be using different versions of data. The only way to make sure everyone is using the same data is to give them shared access to exactly the same data.

Electronic communications enables a dispersed company to have a swift reaction time, by giving managers speedy access to the information necessary to make decisions, and by providing a means of communicating new policies and instructions instantly. In both cases it reduces transit time (that is, from the outside to the nerve centre, and back out again), and thus improves overall reaction time.

Having an international electronic network naturally fosters a global outlook and sharpens skills in coping with a global marketplace. The organization is capable of gathering input from everywhere in the world and looking at patterns in a coherent global picture.

The energy of people in an organization can be dissipated unproductively if the organization is acting in an uncoordinated or conflicting way. The only way to prevent this is to use an electronic network to make the unit a cohesive whole.

## **Software**

And now we come to the third electronic resource we can use: software. Software is in many ways the most significant resource for raising productivity. In many ways, it is also the most difficult to deal with, and there are several sessions at this conference which will delve more deeply into software questions: “Evaluating and Implementing User Productivity Tools”, “Menus, Mice, and Masters in Application Design”, and “The Evolution of the APL Environment: Two Perspectives”.

Essentially, the goal of all software is to process data, hence the name data processing. To be most effective, software must work on both numeric and textual data and gracefully cope with data from a variety of sources. Software must work on a variety of hardware.

Software seems the most personal of the resources we're considering, for software must be suitable to the people who are using it. In the case of an international population of users, this means that software must be available in the native language of their choice. The most important classification of users is according to their level of computer expertise. Computing professionals, computing specialists, and computing amateurs all need powerful software tools.

Computing professionals spend all of their time in data processing. There is a dramatic need for software tools which improve the productivity of such professionals. These people are in the minority but, because they are dedicated to computing, gains in productivity for them have a big payoff.

Computing specialists are those individuals who work in functional areas other than data processing, but who have a particular expertise in or affinity for computing. They often act informally as a resource person for the rest of their department, helping the vast majority of people who are the computer amateurs.

An enormous concern of organizations today is to make the computer amateurs self-sufficient in computing so they can reap the benefits of computing technology. One has only to reflect on the history of the automobile to see the significance of self-sufficiency. At one time, it was felt that there was a limited market for automobiles, because there wouldn't be enough chauffeurs. The invention of the self starter, by making a wide range of people self-sufficient, overturned these expectations. In the same way, software has unlocked the power of computers for huge numbers of people as we have seen with the explosion in the market for PC's.

Many vendors have addressed the needs of the computer professionals. Others have offered software for the amateurs. Unfortunately, the solutions for the two groups have been based on different computer languages or different environments. This means retraining people as they move from amateurs to specialists and on to professionals. It also causes difficulties if a problem proves intractable when tackled by an amateur, using the tools he's been provided with. It then must be passed on to specialists or professionals, and, if they use totally different tools, the process of analysis and solution must be restarted completely from scratch.

Just as there is the risk of inconsistency in the software provided for different users, there is an equal risk of software being available only in certain hardware environments.

There is a risk of inconsistency if different analytic techniques are used in the two different environments. This is particularly true in the age of proliferation of personal computers.

One could say that one piece of innovative software, the spreadsheet, was responsible for the proliferation of PC's and the introduction of computing to vast numbers of users. They are also the furthest removed from the more traditional mainframe-based computing resources of a company. The fact that PC's have a different hardware architecture and are often remote from the mainframe means that the PC environment, while responsible for certain remarkable advantages, is far from an ideal solution.

Software that satisfies the characteristics described above can be instrumental in addressing our business challenges. Data is only data, until software is applied, to manipulate and analyse it, at which point it becomes information.

In order to make quick, well-considered decisions in today's fast paced world, one must be able to analyse many possibilities and alternatives, and to do so swiftly. The decision-making process is further complicated, because it must take into account worldwide factors. Manual efforts could never cope with solving complex models in a short time; the speed and power of software is required to meet these needs.

Software in fact becomes man's most valuable tool, one that extends the power of his brain rather than his hands, and vastly increases his productivity.

## **APPROACHES TO ADDRESS THESE CHALLENGES**

The computer industry has taken several different approaches to bringing these three technologies together, each successful in their own way, but each weak in certain of the areas we've outlined.

### **The Service Bureau**

One approach has been the service bureau. Service bureaus tend to provide good software tools, often combined with a selection of public data. Their efforts are directed to end users, with the intention of helping them with their work, raising their productivity and making them independent of the inhouse data processing bureaucracy. Most offer a network, at least national, and sometimes international. Their failing is their isolation from other software and from data in the corporate data centre.

### **The Data Base Management System**

Another approach is that of the data base management systems. Their goal is to capture all useful data in the organization, make sure it is stored once and only once (thus ensuring data integrity) and that it is independent from isolated applications. There are specialized software tools for working with the data base, for the professional coterie which grew up

surrounding the data base management systems. In order to maintain the integrity of the data, many controls and restrictions have grown up, and the data has become more and more remote from the users in the functional areas.

On the whole, data base management systems have succeeded very well in managing corporate data by bringing it together in one place and by eliminating redundancy. As well, there are good software tools for professionals. Despite these advantages, there are several drawbacks. There's no access to public data, disappointing support for end users and no network support.

### **The Development Centre**

The development centre is in some ways an outgrowth of data base management. Building on the sound foundation of data integrity provided by the data base management, the tools of the development centre can significantly raise the productivity of computer professionals. Huge delays in the traditional data processing department have been a severe worry to organizations, and the software tools of the development centre have helped alleviate this situation. Once again, the major drawback is that the development centre suffers from remoteness from the end users in the organization. And it offers nothing in the way of communications or public data.

### **The PC Revolution**

PC users can also be isolated, and not only when they're geographically remote. Computer amateurs are very productive with little training. Yet, again and again, after the first blush of enthusiastic acceptance, organizations express concern over the lack of integration afforded by isolated PC's. There is a crying need for better communication between PC's, between PC's and the mainframe holding the corporate data, and between PC's and external sources of data.

### **The Information Centre**

The information centre's goal is to provide software tools, training and support for computer amateurs to enable them to become self-sufficient. Its strength is that it has been very successful both in selecting software tools suited for the computer amateur, and in backing that software with good training and support. In most organizations, key corporate data is entrusted exclusively to the corporate data base management system. The information centre offers software tools to extract this data and to make it available to the end users. Thus, while it provides good access to corporate data, the information centre does a poor job of providing access to external data. However, in most organizations, the information centre

is just that—central, that is, based on the data centre. This can leave users who are remote from headquarters poorly serviced.

So the information centre scores very high on software tools for amateurs and good access to internal data, but fails in communications and access to external data. The information centre is very reminiscent of the service bureau, with its emphasis on support and training, and software tools for end users. The information centre combines these assets with access to internal data, while the service bureau offered a network and access to external data. Successful information centres even indulge in active salesmanship to bring the word of self-sufficiency to the consumers, or users. Many of the activities of an Information Centre will be discussed during the panel session “Supporting Information Centre Users”.

### **Introducing A Global Information Centre**

Despite its problems, the Information Centre has successfully brought the benefits of computer technology to a great many people in many organizations. What is lacking is the integration of all the resources technology has to offer. The Information Centre should be extended to provide good communications, access to public data, software for professionals as well as amateurs, and to encompass the capabilities of PC's as well as mainframes. This could be called a Global Information Centre, global geographically because of the network, global in terms of hardware because it extends from PC's to mainframes, and global in yet another aspect because it offers both public and private data.

### **A Global Information Centre: An Example**

To bring reality to this concept, let us consider a fictitious company which has achieved an ideal Global Information Centre.

International Products and Services Association, IPSA, is headquartered in New York, and has 20 product sales offices and 10 consulting services spread over Australia, Canada, France, U.K., Germany, Italy, Finland and Sweden, and 5 manufacturing plants located in the U.S., Holland and Hong Kong.

IPSA has a mainframe located in New York City, with a standard data base management product which provides a consistent source of corporate data. Toronto and London also have local mainframes. All three mainframes are hosts on its international computer network.

All manufacturing plants in the U.S. are connected via terminals to the mainframe in New York. They enter their production plans and inventory levels into the data base, using special programs devised by computer professionals. The logistics department at headquarters compares this data to the sales projections of the sales offices around the world to plan the most efficient distribution of products to where they are needed.

The logistics department uses a powerful software package designed for computer amateurs like themselves. They like the control this approach offers. When IPSA experienced a strike in their U.S. plants last year, American sales offices were supplied from plants in the rest of the world. Because the necessary data and software were ready on the mainframe, a complete new plan was designed and implemented by the logistics department in a matter of hours. The software package is based on the same language that the computer professionals use. When the logistics department comes upon a problem that is beyond their reach or the scope of the software package, they appeal to the professionals in the development centre. The package is so designed that when the package is outgrown, they can gracefully move to the underlying development language.

Consultants in the service division of IPSA prepare many reports on their word processors. When a consulting project involves several locations, reports are transmitted over the network and merged for the final client report. The branches cooperating on a project use electronic mail from these same word processors to discuss the technical details of the project, confirm travel arrangements, and exercise project management control. Each office uses critical path analysis on their part of the project. These individual subnetworks are linked into a common overall network, which is monitored by the service division Vice President in London.

All product sales offices are required to submit forecasts for the annual plan. Each office uses a financial planning package which allows the computer-amateur sales managers to play with the data. Graphic analysis is particularly useful in highlighting trends. Forecasts based on customers sales histories (stored on a local mainframe where there is one) form input to the forecast. Then most sales offices run planning software on their PC (except for Helsinki and Vancouver, who access the same software on the mainframe while they wait for their PC's, and New York which uses the mainframe because it has outgrown the capacity of its local PC). All software is kept in complete synchronization, because as new versions are released on the mainframe, they are also downloaded to all PC's. When discussions of the sales forecast take place within IPSA, everyone knows they are working on the same assumptions.

The sales figures are consolidated for the whole company on the mainframe where they form part of the treasurer's financial model for the whole company. Combining this data with data from a public vendor, which gives currency exchange rate and interest rate trends, he makes plans for capital deployment for the year to come. The final plan is merged directly into the text of his report to the management committee.

Managers in all parts of the company use a convenient user-friendly package to maintain data for a wide variety of applications. These managers were trained in the use of the package by Information Centre support staff. In fact, they first got introduced to PC's through a 'Get Acquainted' session put on by the Information Centre. Some managers use it on a mainframe, some on a PC. If it turns out that others in the company want to use the data, it is uploaded to a common mainframe.

Electronic mail is used to cut across geographic, functional and hierarchical boundaries. There is a mail group consisting of all the sales people in both the products and services division which coordinates sales efforts in different locations with the same client. This group also exchanges market intelligence about clients, prospects and trends. The plant managers also keep in close touch, discussing ideas for improvement of plant productivity, new equipment, and labour management.

The President himself uses electronic mail to publicize company policies and to communicate with employees. He also signs on first thing every morning to get a concise report showing the latest sales and production figures, the cash flow position of the company, and the stock price of the company's stock on the Sydney, Toronto, New York and London stock exchanges.

IPSA's ideal global information centre consists, then, of the following components:

- |                |   |
|----------------|---|
| data           | <ul style="list-style-type: none"><li>• access to timely, accurate internal data</li><li>• access to relevant external data</li></ul>                                       |
| communications | <ul style="list-style-type: none"><li>• a worldwide communications network, linking IPSA's mainframes, personal computers, terminals and word processors</li></ul>          |
| software       | <ul style="list-style-type: none"><li>• integrated software environment for professionals and amateurs</li><li>• consistent software available on mainframe or PC</li></ul> |

### **I. P. Sharp and the Global Information Centre**

During the rest of this conference, you'll be hearing references to I. P. Sharp Associates' products which have been helping customers worldwide to work towards an ideal Global Information Centre. (In fact, it was I. P. Sharp who coined the phrase.) So it's worthwhile spending a few moments now to describe the components of the I. P. Sharp Global Information Centre, so you'll understand these references later.

I. P. Sharp's Global Information Centre starts with data. SHARP APL offers an excellent file system which is suitable for storing large amounts of data and has appropriate security protection built in. Recognizing that many organizations already have significant investment tied up in data stored in data base management systems, SHARP APL provides clean interfaces to this data via auxiliary processors (and work is going on in this area to make them even simpler to use). I. P. Sharp's MAILBOX product has been in use for over a decade and serves to provide electronic mail, links to word processors, and electronic conferencing.

In the field of external, or public, data, I. P. Sharp is the largest vendor of public numeric

data bases in the world, and is a growing source of textual data. This data can be accessed on the timesharing service. Customers running SHARP APL software on their own hardware can also provide access to this data to their inhouse users.

IPSANET is I. P. Sharp's global packet-switched network. It offers access to many computer hosts from over 600 cities around the world, from terminals, personal computers, word processors, and other mainframes. There are links to other public networks, and even to the Telex network.

The final component of the Global Information Centre is a coherent suite of software, based on SHARP APL. APL is the most productive of general-purpose computer languages and significantly increases the productivity of professional programmers. For those computer specialists and amateurs who prefer to make use of powerful application software, it is also available. Again the applications are based on SHARP APL, so that if a package is outgrown, one can gracefully extend it by using SHARP APL. And this language is available on both mainframes and personal computers.

Few organizations have actually attained an ideal Global Information Centre such as that achieved by International Products and Services Association, just as few have implemented all the tools that I. P. Sharp provides toward this end. The sessions "Implementing a Global Information Centre", and "Towards a Global Information Centre" describe efforts to move toward this goal. But, even if not attained, or perhaps not even attainable, the concept of a Global Information Centre provides a unifying strategy which can provide guidance to an organization when choosing the best possible technological directions for their organization.



# **IMPLEMENTING A GLOBAL INFORMATION CENTRE**

Eric B. Iverson, Managing Director, Europe, I. P. Sharp Associates

## **Abstract**

This paper is addressed to the information processing manager, but is relevant to anyone interested in the problems of interfacing and integrating diverse information processing technologies. This paper explains the Global Information Centre concept and shows how to apply the concept to your own organization.



## INTRODUCTION

By 1983, it had become clear at I. P. Sharp that we had built something more than an “information centre”—although we in fact offered an information centre environment and information centre products—and more than a ‘development centre’—although we had environment and products especially suited to rapid development of application software. We realized that our offering was an integrated information/development centre, whose strength lay in its ability to bring different information processing technologies together. *Interfacing* and *integration* were the key words in our products and in our philosophy, but we needed a phrase to sum them our approach and to act as our organizing principle.

The phrase we’ve adopted to cover our concept of the interfacing and integration of all information processing technologies is the *Global Information Centre*, or GIC.

## THE GLOBAL INFORMATION CENTRE

A Global Information Centre is a complete integration of all information processing technologies for an organization, with interfaces appropriate to each technology’s uses.

The ideal GIC is unattainable. But against the ideal you can measure the current state, and with the concept you can plan, implement, and evaluate concrete steps towards improving information processing services in your organization.

You can use the GIC as a framework to think constructively about desktop computers and mainframes; interactive and noninteractive; professionals and nonprofessionals; MIS, DSS, CAD, CAM, and office automation; local and remote; and other previously antagonistic and explosive combinations. It allows you to combine information processing products from different suppliers with different approaches and philosophies into a synergy.

Let’s look at the Global Information Centre in more detail, by discussing its five major dimensions: *people*; *information*; *networks*; *software*; and *hardware*. Key to the GIC concept is keeping a proper balance of attention and resources across the five dimensions of the GIC, and in turn across all the points in each dimension. This balance needs to be analyzed and adjusted continuously.

### People

The most important dimension of a Global Information Centre is *people*: application users, personal computer users, mainframe users, information professing professionals, secretaries, managers, scientists, employees, consultants, suppliers, customers, etc.

In the '60s, attention focused on information processing professionals and their batch systems. In the '70s, attention shifted, frequently with the use of outside timesharing services, to programmers using higher-level languages and managers who were willing to invest a fair amount of effort in using interactive computer services.

Nowadays, attention is focused on the end user. *End user* is not very well defined, but it usually means an interactive terminal user of higher-level applications. People viewed as end users in one organization might be considered information processing professionals in another. The GIC concept asks, Who is *not* an end user?

Everyone is an end user of information processing services; in turn, everyone is a supplier of services to people who are in turn end users. The application programmer is an end user of the system programmer, who is an end user of the operating system designer, who is an end user of the circuit designer, who is an end user of high-level applications.

In a GIC, everyone is a user. Information processing services need to be provided to all the people in a balanced and integrated way.

## Information

The second most important dimension of a Global Information Centre is information—its types, sources, and uses: minutes and memos for secretaries; actuals and budgets for managers; programs and dumps for programmers. Most information is still isolated, under the control of systems, hardware, and networks that are not integrated or interfaced with each other.

To implement a GIC, you need to evaluate what and where the information is, and what steps to take to bring it together. You'll find that the information your organization uses is stored in six different areas:

- Nonautomated: filing cabinets, folders, notebooks, diaries, etc.
- Personal computer diskettes and hard disks
- Local area networks' file services
- Mainframe files
- Mainframe data base management systems
- Online services: I. P. Sharp Associates, Reuters, etc.

A GIC must be able to interface and integrate information from all these areas, so it can flow from those that generate it to those that need it. Each area has its advantages and should be used appropriately; trying to solve the integration problem with today's technologies by simply legislating some of the areas out of existence is not a viable solution. A GIC needs to integrate information by establishing controlled and secure interfaces among these areas.

## **Networks**

People and information are spread around: around an office, around a building, around a country, and around the world. Networks bring people and information together.

A Global Information Centre needs networks to connect people and data within your organization and to connect with people (such as customers, suppliers, and traders) and data outside your organization. You probably have to work with several networks and, consequently, will have to interconnect them. Consequently, you need to consider the particular strengths and weaknesses of local area networks (LANs), private and public X.25 networks, private SNA networks, Telex, IPSANET, etc. A GIC uses each where it is most effective.

## **Software**

Software is concerned with the 'processing' part of information processing. Just having masses of data is of no use at all. You need to be able to process it: to store, retrieve, manipulate, and present it as information. Operating systems, language processors, compilers, development environments, and applications are all software: they all need authors, maintainers, and users; they all have security, integrity, reliability, maintainability concerns; and they should all be concerned with human engineering and appropriate user-friendly interfaces.

We often fall into the trap of thinking of software in different environments as being entirely different, but the similarities are significant. Things learned over the last few decades in the mainframe environment can be usefully applied to personal computers; and things being learned about software in personal computers can be usefully applied in the mainframe. A Global Information Centre must be concerned with the entire range of software.

Software is particularly crucial to a GIC: as well as forming the components (for example, applications) that are to be interfaced and integrated into the GIC, software is how the interfacing and integration of all the elements is achieved. Thus, there are two levels of software within a GIC: the individual software components; and the software that ties the individual components together to form the GIC.

Individual software components need to be evaluated both on how well they perform their function and on how easily and effectively they can be interfaced and integrated with each other. Software without interfaces that allow proper integration is not going to fit effectively in a GIC. Consequently, the GIC concept provides a new measure for evaluating software: Can it be integrated easily and smoothly?

The success of a GIC implementation hinges in part on how quickly and effectively software can be developed to interface and integrate all the required information processing technologies.

## **Hardware**

Hardware is the last dimension of a Global Information Centre—and perhaps the least important. Yet hardware is often treated as the most important, if not only, dimension of information processing.

It seems particularly difficult to keep the right balance in the hardware dimension; and the continuing explosion in the number of hardware products makes it even more difficult. The problem stems in part from the measurability of hardware costs compared to costs in the other dimensions. The cost of a faster mainframe to reduce response time is easy to calculate and understand. The cost of not reducing response time, although documented in studies, is much too difficult to calculate and understand. Some organizations react to tangible hardware costs by investing in what they can measure, but ignoring the less tangible. Other organizations react by clamping down tight controls on these tangible, measurable costs. Both reactions result in serious imbalances that make it impossible to have an effective GIC. Spending too much on hardware is a waste; spending too little results in money spent elsewhere being wasted.

It should be clear that in a GIC money is not spent on hardware for the sake of the hardware, but rather it is spent to maximize the benefit from the total investment in all five dimensions of the GIC.

## **INTERFACES AND INTEGRATION**

The key to implementing a Global Information Centre is to use the interfaces of the individual elements to integrate them into a whole that is greater than the sum of its parts. To do this requires elements with good interfaces, the ability to develop or improve interfaces where they are lacking, and the ability then to integrate the elements. This is a continuing and interactive process.

Each element in a GIC—for example, an application or a special graphics device—should be viewed as a building block to be combined with other building blocks to solve problems in new ways. An application can not only be used directly but also as part of a higher-level application.

Eventually integrating what are now isolated elements is essential to the GIC concept; and recognizing this is a general requirement of implementing a GIC.

The same analysis and methods that now apply at lower programming levels (modular design, functional design, clean interfaces, use of common utilities, etc.) should also be applied at the top levels. It should be possible to build new, higher-level applications out of existing applications, the same way programmers build applications out of existing program modules.

This is often not the case. The external appearance of high-level applications often contradicts their internal structure. Internally, they usually use proved techniques involving modular and functional designs; externally, they usually do not. This is not too surprising as these are new considerations and there is an almost complete lack of standards. In addition, applications put great emphasis on the quality and variety of interfaces with the end user; but not much attention is paid to the interfaces required by programmers in order to integrate these applications with each other. This is likely to be the case for some time to come, and it means that the GIC needs to be particularly concerned with how to handle applications with inadequate or even nonexistent interfaces.

## **SHARP APL**

I'd now like to address I. P. Sharp's technologies, and how they fit along the dimensions of a Global Information Centre. The next two sections summarize our view of where we—as a company and as a manufacturer of software—fit in the evolution of the GIC.

Where does SHARP APL fit into the GIC? SHARP APL provides an information centre environment, and information centre products like VIEWPOINT and MAILBOX. It provides a development centre environment [Goldsmith], and the ability to develop new applications effectively and quickly with particular emphasis on user-friendly interfaces. It provides access to the international communication network IPSANET.

The real value of SHARP APL in a GIC is that it has efficient and effective basic interfaces to all other elements in the GIC; and it has a language and development environment that allow these interfaces to be used in making the integrations required by a GIC.

A GIC requires a great deal of interfacing and integrating. Unfortunately, the elements of the GIC don't usually fit together as easily as the bricks of a Lego set; they require a mortar of software in order to build up the GIC. And because the GIC, in a sense, is built in an earthquake zone of major and sudden changes, it requires a mortar that is flexible, maintainable, and even capable of quick rebuilding. A GIC needs the ability to use interfaces and to integrate radically different elements quickly, efficiently, and reliably. This is exactly the kind of problem for which SHARP APL is ideally suited. SHARP APL can tie all the disparate information processing technologies together into an effective GIC.

The GIC and SHARP APL were essentially made for each other.

## **I. P. SHARP ASSOCIATES**

Where does I. P. Sharp Associates fit into the Global Information Centre?

We've been in the business of developing and exploiting information technologies for twenty years. Our people are familiar with the GIC concept and have experience in applying it; they are expert in specific areas of information processing, but are also able to see the complete picture.

In our twenty years we've built up a great deal of experience in dealing with all five dimensions of a GIC. In the people dimension, we are familiar with the information processing concerns of everyone—professional and nonprofessional alike. In the information dimension, we've done pioneering work in several specific areas and have experience in all areas. In the network dimension, we are particularly strong: we've developed and operate a large international packet-switching network which interconnects with X.25 networks, Telex, and several hosts; as well, we operate our own SNA network and are involved in LANs. The software dimension is where we have the most extensive experience; and, as required in a GIC, we have software experience at all points in the dimension: from system level to very high level, on micros, minis, and mainframes. Similarly, we have experience along the hardware dimension.

Our experience is not only theoretical or second-hand. Since 1969 we've operated an information/development centre, which has grown and evolved into a GIC. As well, we've been in the business of helping other organizations set up and run their own information and development centres since 1976. We are now involved with more than forty GICs running SHARP APL around the world. Not all of these GICs fully exploit all the dimensions of a GIC, but because what they have is a consistent part of a larger picture they have a clear growth path towards a complete GIC.

We have a philosophy—the GIC concept—and we have the people and products to help you implement it. The concept is not an abstract one; we use it ourselves in running our own GIC and we have worked with it in helping other organizations set up their GIC.

## **THE INFORMATION PROCESSING DEPARTMENT**

The view of information processing in a Global Information Centre not only recognizes the major role that needs to be played by an organization's information processing department, but also helps define it. The information processing department is not only responsible for providing information processing services, but also for implementing and integrating these services into a GIC. The department needs to be aware of and involved in all the dimensions of a GIC.

Throughout the brief history of information processing, a recurring theme has been the introduction of new technologies outside the scope of the information processing department that are eventually brought into that scope. Initially the department was concerned only with professionals and batch production systems on mainframes. It was unwilling or unable to cope with the requirements of interactive services, so these services grew up completely independently, on outside service bureaus that specialized in them. Much of this has now



been brought into the scope of the information processing department and is considered not only a normal part of the department, but as a key part. Similarly, distributed processing started out independently of any central department, but is now generally seen as just another aspect of services provided by an information processing department.

New challenges now come from the explosion in personal computers and office automation. Again, most of this is happening outside of the information processing department, and little or no attention is being paid to the eventual and inevitable problems of interfaces and integration. The problems presented by interactive services and distributed processing were trivial in comparison. Although the potential is commensurate with the challenge, the information processing department views this explosion in information processing technologies outside of its control with alarm, because it knows that eventually it will have to bring the whole thing together.

The Information Processing Department needs a GIC view of the world in order to make sense of it. They need to start now in better integrating what they have, and in bringing in the parts that are springing up beyond their control.

## **YOUR GLOBAL INFORMATION CENTRE**

Setting up a Global Information Centre is not an easy, one-time process. It is a complex procedure that requires a lot of analysis and work that has to be done on a continuing basis as the situation evolves.

First, you need to understand the GIC concept at an abstract level and then you need to make it more concrete by studying how other organizations have applied it. It is essential to keep current with what is happening in both new and old technologies by using the literature, conferences, and the consultants and sales, marketing, and technical representatives of companies in the industry.

At the same time, you need to study your own organization in order to see what is relevant and how to apply it. You need to understand how your organization compares with others in the marketplace of information processing technologies. Is information processing central to your business or peripheral? Is your organization pioneering or conservative? Is it fast-moving and flexible, or more methodical and fixed? The plan for implementing your GIC has to fit your organization.

You need to evaluate your current situation. Measured against the ideal GIC, where are your strengths and weaknesses? Where are you out of balance? Where are the major opportunities for benefits? What parts of a GIC do you already have? What parts do you need to add most? What things do you have that won't fit? What is possible within the reality of the organization?

With this information, you can then plan how to implement your GIC. This plan has to keep an eye on the long-term GIC goals, make effective use of available technologies and products, fit in with the realities and requirements of your organization, and move you from your current situation towards the provision of better information processing services to all parts of the organization. This is an interactive process that should never stop.

## **A First Step**

Installing the SHARP APL base system is a significant step towards building your Global Information Centre. In the past, it was difficult; our products were bundled together into a very large package, and were geared towards only the largest and heaviest users. This meant that the first step was not so much a step as a staircase; the costs and resource requirements were too high for many organizations.

To make the first step easier, the SHARP APL has had its own interfaces reworked; it has been unbundled into logical components that are available separately as building blocks. In addition, work has been done, and continues to be done, to reduce to a minimum the resource requirements for entry-level systems. Combined with these technical developments, prices have been reduced as we've started to step up a marketing and sales effort to move more into the large market appropriate to the products and the market's requirements.

Now, at very low costs, you can easily install the base SHARP APL system on your mainframe with minimal commitment of hardware and people resources. Once this base system is installed, you can plan the subsequent steps as appropriate in building your GIC.

## **More Steps**

Once SHARP APL is installed, you have opened up many possibilities for your Global Information Centre.

An easy and obvious step is to add some of the information centre products like VIEWPOINT, MAILBOX, and MAGIC. These products give significant, immediate benefit and fit in well with most GIC plans.

If you require the development of custom software to solve problems not dealt with by packaged software, then you will want to install development centre products like LOGOS and APE.

You can provide worldwide access to your systems by connecting them to IPSANET. This gives secure, convenient, and efficient access to your mainframe from around the world. IPSANET is in turn interfaced to public X.25 networks and the Telex network giving even further reach to your systems. IPSANET links can be used by unintelligent terminals, personal

computers, workstations, minicomputers, and even other host mainframes. You can connect your mainframe to IPSANET with an IBM 3705 communications controller running either dedicated IPSANET software or standard IBM software with an X.25 interface.

Access to I. P. Sharp Associates' online services can be provided via a gateway using an IPSANET link. This allows, under program control, real-time selection of data from the public data bases, and downloading of this data to your own systems for processing. The data of our online system is thus added to the total repertoire of data available to your user.

Access to other systems can be provided by adding the appropriate auxiliary processors and associated software. The basic tools exist to interface and integrate any system in your mainframe environment.

Making more effective use of personal computers by integrating them into your GIC is one of the most important problems to be tackling now. This is a new area with explosive growth and few standards. The problems are considerable, but the potential benefits are enormous. The first goal should be to provide all the personal computers with the basic ability to connect into your other systems and to upload and download data. This can be done via asynchronous networks, SNA networks, or LAN gateways. We have considerable experience in these areas as well as several key products. One new product is a general workstation designed for the IBM Personal Computer. Another is the high-performance version of SHARP APL for the IBM Personal Computer XT/370.

## **SUMMARY**

Information processing has become a very complex business. An overall philosophy or view is required to reduce this complexity to manageable forms, and to allow the effective planning and implementation of information processing services. The concept of the Global Information Centre provides this necessary view, and products and expertise from I. P. Sharp Associates, combined with products and expertise of other information processing professionals, allow the concept to become a reality for your organization.

## **REFERENCES**

Goldsmith, Leslie H., David Allen, Mark Dempsey, and Kevin L. Harrell, "LOGOS: An APL Programming Environment" in this volume.



# **EVALUATING AND IMPLEMENTING USER PRODUCTIVITY TOOLS**

Robert Metzger, I.P. Sharp Associates

## **Abstract**

Information centres exist primarily to provide user productivity tools. The best of these tools improve productivity by providing integrated data and non-procedural processing. This paper explores these two key concepts, so that products may be evaluated accordingly. It also suggests ways of implementing them in SHARP APL.



## **INTRODUCTION**

Productivity is an essential concern of anyone managing an organization. Organizations purchase computer hardware and software to improve productivity. Such computer systems must enable their employees to do one or both of the following:

1. Provide the same goods or services at a lower cost, or
2. Provide superior goods or services at the same cost

Almost all computer system vendors claim that their products improve productivity. How can a potential purchaser of such products evaluate such claims? Just who are the people who will become more productive?

There are two kinds of people who use computers:

1. Those who use computer systems to build other computer systems
2. Those who use computer systems to accomplish work related to the goals of the organization

Programmers fall into the first category, 'end users' in the second. For the purpose of this paper, we'll refer to these groups as 'developers' and 'users'. As the title of the paper suggests, we will only consider software tools which enhance the productivity of 'users'.

This paper is divided into two parts. The first part gives some general criteria for evaluating the usefulness of user productivity tools. Anyone who has spent a few hours working at a computer terminal or with a personal computer should be able to understand this section and apply its criteria. The second section gives specific suggestions on how to implement user productivity tools in APL. It assumes that the reader can program in APL, and is familiar with video display terminals or personal computers. The appendices include a checklist for evaluating user productivity tools, and APL programs that can be used to implement them.

## **ENHANCING USER PRODUCTIVITY**

Today's user tools can improve productivity in several ways. This paper focusses on two of them. The first way is to provide integrated data. The second is to provide non-procedural processing. These two categories come from the nature of data processing, and even from language itself. In a natural language like English, the primary categories are nouns (objects) and verbs (actions). In the phrase 'data processing', we have 'data' and 'processing'.

While the division is quite natural, you might wonder why 'integrated' data and 'non-procedural' processing make users more productive. This section of the paper answers that question.

## Integrated Data

What do we mean by integrated data? Data may reside in many physical locations in many different formats. Integrated data appears (to the user) to be in one location in compatible formats.

Why does integrated data make users more productive? First of all, it is more convenient to work with. Every minute a user spends looking for a particular file is a minute he could have spent solving a problem. If the data appears to be all in one place, he doesn't have to log on to one system after another, vainly seeking a file whose name he can't remember.

Integrated data also makes users more productive because it ensures completeness. A business analysis may be totally invalid because a user is not able to get to a crucial set of values stored on file. This happens all too often when data is stored in dozens of incompatible file formats. If all the necessary data is available, the user will do his job better.

Finally, integrated data makes users more productive because it provides consistency. It isn't enough to just store all data in the same file format. If some of the data is weekly and other data is quarterly, and the software provides no conversion between the two, it really isn't integrated data. The consistency of the **content** is just as important as the consistency of the **form**. If the content isn't consistent, it might as well not be available. Otherwise, the user must do one of the following:

1. manually convert and re-enter the data
2. get a developer to convert the data
3. do without the data.

None of these 'solutions' is acceptable.

These are just some of the many reasons why integrated data makes users more productive. Of course, the integration will probably be done by software. What is important is the appearance of integration, not the particular means used to provide it.

## Non-procedural Processing

Why does non-procedural processing make users more productive? We must first consider what 'non-procedural' means. The following is an everyday example of a non-procedural statement:

I'd like a chocolate cake with chocolate frosting.

There are several procedural statements which will produce the same results. These include a from-scratch recipe, or the directions on the back of a package of cake mix, or directions to the nearest bakery. Each contains a series of steps that must be followed precisely if I



am to be served a delicious dessert. Clearly, some of the alternatives require less work than others. Using a mix is easier than baking from scratch. But in each case, the recipe (algorithm) contains a sequence of steps that involve iteration, alternation, and hierarchy. In the case of making a cake, examples include:

stir until thoroughly mixed  
if the frosting isn't moist enough, add a teaspoon of milk, and  
for the proper way to crack eggs, see *The Joy of Cooking*.

The non-procedural statement describes **what** I want, the procedural statements describe **how** to provide it.

So why does describing 'what' I want, instead of 'how' to do it, make users more productive? First, it simply takes less effort to describe what you want rather than how to get it. So, non-procedural processing makes people more productive by enabling them to do more work in the same amount of time. Second, it reduces the amount of training needed for people to use a computer productively. There are lots of people who have the innate ability to program who don't have the time to learn how. They're too busy doing their real job. Finally, it makes the benefits of using a computer available to a wider audience. You don't have to be a mechanic to drive a car, and you shouldn't have to be a programmer to use a computer.

### **Distributed Programming**

User productivity tools do reduce the applications backlog faced by DP staff. However, this saving does have a cost attached to it, a cost seldom discussed by software vendors. This is the phenomena of 'distributed programming'. Because such tools make more people capable of using the computer, requests for systems development that were formerly handled by the DP department are now done directly by users. More hands are working, and thus more work is getting done. This would have been true even if the users had been coding in COBOL. What the user productivity tool provides is leverage.

User department managers must remember some important facts. If they are going to provide user productivity tools to their employees, those employees will not be available for department work for some percentage of their work week. The department manager has converted the cost of paying the DP department to do the work in COBOL (or the cost of not doing it at all), into the cost of having his own people do it. The exchange rate is favourable when his own people are using a good software tool, but there's no such thing as a free lunch.

## **INTEGRATED DATA**

What kinds of data can be integrated? There are at least three levels of integration:

1. integrated files
2. integrated data bases
3. integrated machines

A complete user productivity tool must provide integration at all three levels.

What's the difference between a file and a data base? For the purposes of this discussion, a file is a collection of data whose appearance to the user closely resembles the actual way it is stored. In contrast, a data base is a collection of data whose appearance to the user may radically differ from the way it is stored. The 'appearance' includes the sequence of the data items, the way they are grouped, and the representation of a single data item.

### **Integrated Files**

How should files be integrated by a user productivity tool? First of all, users should be able to export and import data from any of the other flat file structures in common use on the same computer system. By export and import, we mean copy selected fields and records. In an IBM MVS environment, a user productivity tool should be able to import data from OS and VSAM datasets, and export data from the productivity tool to these datasets.

Secondly, users should be able to import and export data from other user productivity tools that are in common use on the same computer system. For example, a user productivity tool running in the SHARP APL environment should certainly be able to retrieve data from standard APL component files, in which data from custom-written applications is stored.

Thirdly, users must be able to link records which have common key fields in a manner similar to the relational join and project. This is important to encourage users to share data. If a user knows he can easily append to his data some information his colleague is collecting, he is far more likely to avoid storing the same data in his own file.

### **Integrated Data Bases**

How should data bases be integrated to improve user productivity? Simply owning a data base management system (DBMS) is no guarantee that data is integrated. The query language provided with the DBMS probably doesn't provide the means to work with data from other DBMS, or flat files either. So, those who use it can integrate any data it controls, but are isolated from files outside of its domain. This nullifies most of the meaning of integrated data.

The user productivity tool must provide the links to a variety of DBMS, but not be the subordinate of any of them. It should be capable of extracting data from DBMS which follow any of the three data base models:

1. hierarchical (i.e. IMS)
2. network (i.e. IDMS)
3. relational (i.e. DB2)

If data exists, the user should be able to get at it, wherever it is, with minimum effort.

Users will be more productive if they all share a common model of the data they are using and exchanging. This common outlook, and the simplicity of the concept of tables, makes the relational approach the preferred one for user productivity tools.

A subject that usually gets lumped in with data bases is data dictionaries. They are a very important way to improve the effectiveness of computer use. A user who has a need should be able to locate the data available to meet that need. While the data processing details in a dictionary will be maintained by the systems staff, it should be easy enough to use that a novice can find what he is looking for. The computer details should be artfully hidden from those who don't need to see them.

### **Integrated Machines**

Going back to our original definition of integrated data, it is important to remember that integration extends beyond files and data bases. In today's information environment, it is becoming increasingly important that data be accessible on computers other than the one on which it is located. The emergence of network technology and the widespread use of personal computers make this both necessary and possible.

At the simplest level, a user must be able to download data to his personal computer, work with it, and then send it back up to the mainframe. Any user productivity tool for use on the mainframe must contain the following capabilities:

1. download an extract of a file to a personal computer
2. upload data from a personal computer to be included in a mainframe file
3. use the personal computer as an intelligent workstation over a network

Of course, it would be really nice if the exact same software was available on both the personal computer and the mainframe. The next generation of personal computers, such as the IBM XT370, have the power to provide such compatibility. These new machines will make users more productive as they have to spend less time learning incompatible products.

Many companies have several large computer systems. Often the work on these machines is segregated into categories like the following:

1. production vs. development
2. operational vs. user computing
3. data base vs. non-data base

The problem with these separations is that they tend to make user computing less productive. In most situations, if data is on one machine, it is not accessible on another machine. If so, that means that the users must do their computing without access to the very data that controls the key operations of their organization. Or, they must wait for time-consuming manual transfers of data copied to magnetic tape. If data base management systems reside on a separate machine, a complete user productivity tool will provide the user with an easy way to transfer data so his computations are based on reality.

### **NON-PROCEDURAL PROCESSING**

There are four major questions that must be asked about any system that claims to provide non-procedural processing:

- Is the processing reactive or proactive?
- Is the processing segmented or apertural?
- How does the system process natural language input?
- How does the system use special input/output devices?

The various advances or fads (depending on your perspective) in user productivity tools can all be understood from this framework.

### **REACTIVE VS. PROACTIVE PROCESSING**

What is the difference between reactive and proactive processing? In reactive processing, the software asks and the user answers. In proactive processing, the user commands and the software responds. Typically, reactive applications consist of menus to choose from, and forms to fill in. Whenever the user has an opportunity to enter information, or do some sort of activity, the alternatives possible are extremely limited.

When working with most systems of hierarchical menus, the user has three options:

1. select an item on the menu
2. ask for more information (help) about an item on the menu
3. not select any item—thus returning to the previous menu

When working with a system that requires forms-filling, the user has two options for each field:

1. fill it in, or
2. leave it blank (if the software allows that)

Such menu/forms systems are what people have traditionally called interactive computing. Many are quite successful, but they are designed for a particular type of user. Such systems presume that the user needs to be led by the software. This may in fact be true, if the user is afraid of computers, or isn't acquainted with the program, or doesn't want to take the time to learn how to use it. As personal computers become as commonplace as televisions, the first group is diminishing. The second group will always be around. The third group can be reduced in size by developing software which provides transferable knowledge—if you learn vendor X's package 1, you will know most of what you need to know to operate package 2 as well. As software developers become sophisticated, this will become more common.

### **Proactive Processing**

In proactive processing the user commands, and the software responds. Reactive processing has its proper place, but it is most interesting to note that the most popular software packages in today's market are proactive. The most obvious example is the spreadsheet. At any time, the user can:

1. type data into a cell
2. type a mnemonic command
3. use the cursor and function keys.

The spreadsheet responds to the commands the user enters, and the possible choices are anything but limited.

Personal computers are creating a whole new class of sophisticated users. Anyone who has used a spreadsheet on a personal computer is going to be quite reluctant to go back to reactive processing on a mainframe. We are going to see an increasing distinction between the 'passive' and 'active' user. The passive user is content to run reports defined by someone else, or enter data into forms designed by someone else. The active user wants control over the computer he uses. He is a do-it-yourself type who is only satisfied with proactive software. The software tools that will continue to bring productivity gains will be increasingly powerful proactive applications. The paradox is that they must also be capable of being made into reactive applications to accommodate that large body of passive users who are not going to go away.

The two major components of proactive processing that aren't really used in reactive processing are a command language and function keys. A reactive system has no command language at all, and its use of function keys is limited mostly to HELP and STOP. How should these components be used to maximize user productivity?

## **Command Languages**

Consistency is an essential quality. If an application provides a consistent command language and function key definitions, the time required to learn and use the application will be decreased.

How can a command language be consistent? First, spelling of keywords should be consistent. If the command to display data is spelled **PRINT** in one module, and **SHOW** in another, confusion will reign. If abbreviations are supported, they should be done by a standard scheme (acronym, truncation, or condensation). If the system doesn't support a full abbreviation scheme, but only shortened aliases for words of a certain length, all long words should have aliases.

Second, the command language should have a consistent syntax. If action words precede object words, they should always do so. If operations are normally an action-object pair, separated by blanks, they shouldn't also be sometimes separated by a special character. If parentheses are used for grouping, their scope should be the same. If commands are punctuated in a special way, they should always require the same punctuation.

Third, the semantics of the command language should be consistent. If a keyword has one meaning in one context, it shouldn't have a totally different meaning in a different context. If synonyms or string substitution are provided, they shouldn't alter the meaning of the statement.

## **Function Keys**

Consistency is the most important requirement for the use of function keys as well. First, a given command should always reside on the same key. The system designer should make full use of habitual behaviour. Good work habits, cultivated by the designer, make the user able to work more quickly, since less thought is required. If the user habitually presses a function key to save his work, but finds in certain activities it does something else, he will become thoroughly frustrated.

Second, if potentially dangerous commands are provided by function key, they should be protected. Such commands mostly involve erasing or overwriting data. If they aren't protected, the slip of a finger can eliminate valuable work. One protection scheme requires a user to press dangerous keys twice. Another one displays a message and requires the user to confirm before taking action.

Third, careful consideration should be given to which commands are available from function keys. In general, the most commonly used ones are the most desirable. This strategy, of course, improves the productivity of the user by reducing keystrokes.

Finally, the layout of the keypad should be considered. Will most of the users be familiar with a given layout because they use another package? Don't make them learn another layout. Are the potentially dangerous keys right next to keys that would protect or help the user? For example, putting ERASE next to SAVE or HELP is not a good idea. Is the full power of the keyboard exploited? Some keys that aren't labelled 'function' can serve that purpose, if handled appropriately. The ESC key on an ASCII keyboard, or the DUP and FIELD MARK keys on an IBM 327X can be exploited quite effectively.

## **SEGMENTED AND APERTURAL SYSTEMS**

What is the difference between a segmented and an apertural view of functionality or data? In the segmented approach, the developer defines the user's view of both the data manipulated by the system, and the functions of the system itself. The developer carves up the segments, and the user lives with the result. In the apertural approach, the user can dynamically define his view of the data and functionality, as if he were aiming an aperture at parts of the system.

The segmented approach is the one traditionally used in data processing. The most typical example of this approach to functionality is the hierarchical menu structure. The user must traverse this menu to get to the desired task. Such menu systems can be made relatively painless to use. You can see their weak points when an activity that doesn't really belong with any other branch, or belongs with several, is found. The program developer puts it in the hierarchy where he sees fit, and lots of users spend their time going up and down menu trees.

The segmenting of functionality goes much farther than menus, however. If the phrase 'you can't do that here' keeps ringing in your ears, you know that the developer has limited capabilities when they might logically apply elsewhere.

Relics of the past also segment functionality in unnatural ways. The distinction between query and update in an apertural system is meaningless. A user almost always needs to see information before changing it. In an apertural system, the user can see the data he wants to change on his screen, and type right on top of it. He doesn't need to first query the system to list the data, and then go somewhere else to change it. Another useless distinction is between report generation and file updating. If a report writing program can select data and do calculations, it should be able to display the results, or store them in a file in a form suitable for further manipulation. The only difference in the results is the way they are stored—as a text or as records that selection and mathematical operations can be performed on.

Of course, data is segmented by developers as well. The most common form of this design is when the developer, not the owner of the data, limits the access others may have to the data. If such access can't be maintained online by the owner, intervention by the developer to handle changes becomes inevitable. Other types of segmenting include separation by organizational level, and presentation of summary or detailed information only.

## Apertural Systems

The apertural approach lets the user segment the functionality of the system and the data as much as possible. Rather than have limited selections of functions in menus, all commands are available simultaneously. The user chooses with a command language or functions keys. He uses these to focus the aperture of the system on the function or data he needs.

The apertural approach is most easily understood when applied to data. It requires a fullscreen video terminal to implement. At all times the screen contains a portion of the data being viewed, as defined by the user. Three types of operations are used to define the view:

1. Scroll—move the aperture (window) over the data. The aperture should be movable both horizontally and vertically.
2. Slice—select specific pieces of data and move them to the aperture. Once again, slicing should be available both horizontally and vertically.
3. Shuffle—re-order the data values so that new information becomes visible in the aperture. The shuffling can be done by sorting, moving, or holding specific items.

Of course, the spreadsheet is the archetypical apertural program. You can't see the whole 'sheet', but you can scroll about to view all of its cells in turn. The integrated spreadsheet/datafile manager packages provide slicing and shuffling as well.

As with the question of reactive vs. proactive processing, the choice between segmented and apertural applications depends on the type of user. The passive user will probably be willing to look at data in a pre-defined report. He will also be quite willing to have his view of the functionality severely limited. Such static, pre-defined views of software and data can actually make some users **more** productive. This can happen if they don't have the time or ability to learn an apertural system. But the wise developer builds applications from an underlying apertural system. This approach makes the developer more productive than coding an application from scratch in a procedural language. The underlying apertural system can be made available to the user should he outgrow the segmented approach.

When the user is doing data entry or update, several additional features are required for an apertural system that only a full screen video device provides. The user must be able to change values by typing over them. This implies instant update of the display, and cursor control keys as well. It is very helpful if the display attributes of the screen show the various states of data on the screen, i.e. original, changed, deleted, etc. Finally, editing keys (insert/delete) can enable a user to make changes much quicker than when using a teletype terminal.



## **Design Principles**

There are several important principles that should be observed when designing an apertural system. First, the user must be able to see what he is working with. This is the whole point of apertural systems. The reason that they make users more productive is that the user doesn't have to remember as much. What he needs to know is right on the screen in front of him.

What exactly should be displayed? If the program has different sections or modules, the section (or screen) should always be identified. If the user is working with a data file, its identifier should also be on the screen. If the function key definitions are absolutely identical in every part of the system, use a keyboard template. Otherwise, display the definitions on the screen. If the user must select from a finite list of choices, they should all be on the screen, preferably with an accompanying description. Of course, as much data as possible should be displayed on the screen.

The second principle is that the user should be provided with an appropriate level of detail, and that he should be able to change that level. For example, if a user is viewing a list of fields available in a file, initially he probably won't need to know how they are represented internally. He just wants to know their names and descriptions. Later on, when he wants to add a new field, he should be able to define the internal representation if he desires.

One way to provide the appropriate level of detail is to omit things. Another way is to summarize the information. For example, if a user is viewing a list of files he owns, it is usually sufficient to indicate whether a file is shared or not with a simple YES or NO. If he wants to know who has access to the file, or what type of access they have, he should be able to expand on that simple answer. This is a concept referred to as 'zooming'—like zooming in with a camera. The user may be able to peel off several layers of summary by successively zooming in on a particular attribute. Typically this is done by moving the cursor to the data cell of interest, and pressing a function key. A new screen is then displayed containing the detailed information.

## **Help Screens**

The third principle for designing apertural systems is that the online helps should be active, not inert. Inert helps are the traditional approach. They are just blocks of text on the screen, explaining what the user should do. This approach is used because people have considered helps to be a part of the documentation of the system. In an apertural system, they are a vital part of the software itself.

Active helps take a variety of forms. First, the user must be able to get help both with a function key and with a command typed in. Both general and specific helps should be available using both methods. Some areas of the screen will require special helps. For example, if the cursor is on the command line, and the user presses the help function key, the result should explain the valid commands. If the cursor rests on a data cell, the result

should explain how to view and change data cells. If the user types the command **HELP**, a general information screen should be provided. If the user types **HELP** and some topic, the system should be capable of checking the topic against a list of topics and displaying the appropriate screen. A really helpful system will also maintain a list of synonyms for its topic list. As a last resort, it can even string search the available help screen texts looking for a reference to the topic in question.

Active helps can go one step further, and actually help the user to operate the system. This can be implemented in places where the user is presented with a choice. If he doesn't know which he wants, he should be able to ask for help. The help screen should explain the choices, and provide a space to select the option right next to the description. When the user returns from the help screen, the selection is automatically copied to the original screen he was working on.

This approach can even be taken one step further. The helps for a 4th generation language can be menus or forms which explain the most fundamental commands. The user selects commands from the menus, or fills in the blanks on the forms. The completed set of statements is shown to the user as an aid to learning. This type of help makes it possible for the user to access the power of the 4GL without having to remember the keywords or the syntax.

## **NATURAL LANGUAGE PROCESSING**

Natural language input is one of the current fads in user productivity tools. A number of companies are claiming to have such capabilities, but all that most of them have is clever ad copywriters. A user productivity tool may provide a 4th generation language and still not have true natural language processing. In fact, most don't.

### **Fourth Generation Languages**

What exactly is a fourth generation language? It is distinguished from third generation languages like **FORTRAN** or **COBOL** by the fact that the user specifies **what** he wants done, rather than **how** to do it. There are several other subsidiary differences:

1. The syntax is much more flexible
2. Common English words with intuitively "obvious" meanings are used
3. The order in which the statements are processed is irrelevant
4. Phrases can be combined in meaningful ways

The reason some people confuse 4GL's with natural language processing is that they look like English, at least on the surface. They use English vocabulary, and a syntax reminiscent of English. In reality, they are pseudo-English. This is not a put-down, just a realistic assessment.

How do 4GL's differ from true natural language? First, they are based on keyword recognition. One specific word has one meaning. Even if the language allows for synonym recognition, one word cannot be the synonym of two keywords. That's the way people use natural languages, but a 4GL can't handle such ambiguity.

Second, they really have a quite rigid syntax. In English, we have very few rules governing word order. We put verbs before nouns to ask questions, and omit the subject to create commands. In a 4GL, everything looks like a command, and each statement must begin with a command word. The command 'verbs' are actually English verbs and prepositions, which is another difference between a 4GL and English.

Most importantly, a 4GL has no real understanding of the meaning of what it is processing. When we tell a human something, we feel that they have understood if they can correctly answer questions about what we have just told them. 4GL's don't provide any means to inquire about their understanding of what we have said. Moreover, they have no capability to infer things from the commands we have given.

The fact that 4GL's do not process natural language does not diminish their usefulness. It merely puts it in proper perspective. The use of English words isn't what enables them to make users more productive. It is the fact that they provide non-procedural processing in a proactive environment.

### **True Natural Language Processing**

There is great potential in natural language processing, even though it has become an advertising buzzword. There are three questions that must be answered if you are really going to understand the significance of this concept:

1. How does real natural language processing work?
2. Why do people want natural language processing?
3. What applications will benefit from true natural language processing?

True natural language processing is adaptive, rational, and passes the Turing test. An adaptive program improves its performance over time. In the case of natural language processing, the program must adapt to the patterns of expression of users. For example, if someone uses professional jargon in interaction with a program that provides natural language processing, it must be able to inquire about the meaning of the jargon and then remember its meaning thereafter.

A rational program is one that can explain what you have told it and what it has said or done. It can infer things from what you have said, and it can tell you why it knows what it knows. For example, if a program interprets a query request in a different way than the user expected, he should be able to ask for an explanation of the reasons the program acted as it did.

The Turing test is simply this: can a knowledgeable user tell whether he is interacting with a program or with a human? If he can't tell, the program passes the Turing test of intelligence. In the case of natural language processing, this means that the program must handle the full range of normal human communication, including such difficult problems as pronoun references, ellipses, conjunctions, and noun modifiers.

### **Natural Language Applications**

The real reasons people want natural language processing are seldom stated. The oft-stated answer is that it will make it easier for people to use computers. This answer ignores a more fundamental problem—the keyboard. The very people who aren't familiar with computers are the same ones that don't type, or don't type well. Until voice recognition systems are capable of responding instantaneously to a large population of speakers with an unlimited vocabulary, those few user productivity tools that allow natural language input will be quite restricted in their effectiveness. It will probably take another five years before this problem is solved.

The real reason that people want natural language processing is that they want the computer to know more. Every human has an unimaginably large store of information about the real world that allows him to understand natural language. Programs that process natural language will be easier to use because they require the user to say less. They will require less information from the user because the programs will know more.

What sorts of applications will user productivity tools that process natural language be really useful for? Probably not the principal one being sold today, which is data base query. The real potential lies in programs that can assimilate, understand, and summarize large quantities of text. This brings us back to our theme of integrated data. There is an immense amount of unorganized textual information that business organizations could make good use of if only they could have a program summarize it the way a report writer summarizes numeric data into nice reports. As it stands, the information is just as useless as large quantities of raw statistics.

Some examples demonstrate the potential impact. How much time would the executives of your organization save if all interoffice memos were sent through an electronic mail system which was capable of summarizing pages of text into a few lines? They could keep on top of everything, while actually reading the detailed texts of only the urgent items. How much money would a manufacturing organization save if detailed call reports by service engineers were available online through a program that summarized their contents? When a new product was introduced, the development engineers would be able to quickly spot patterns in the problem reports, even if hundreds of service reports were filed daily. How many risky investments would be avoided by financial organizations if they had services available that summarized news stories from around the world from dozens of sources every day? They could spot investment opportunities and danger areas much more effectively.

True natural language processing is available with only a couple of user productivity tools. The pseudo-English approach of most such products is quite sufficient at the moment. Most companies are still moving from traditional data processing to non-procedural processing, and aren't ready to make use of artificial intelligence products yet. During the second half of the 1980s, natural language processing will make users more productive, but this will come primarily through handling new types of information, rather than in simpler user interfaces.

## **INPUT/OUTPUT DEVICES**

A user productivity tool can be made more effective through the use of special input and output devices. Perhaps the most common of these is the graphics display or flatbed plotter. Graphics can make information come alive. All user productivity tools should provide the option of displaying reports in graphical form.

The second most common input/output device is the personal computer. There are several reasons why personal computers networked with a user productivity tool on a mainframe can make users more productive. First, the redundant processing units minimize time users waste waiting for a system that is 'down'. If the mainframe isn't working, the user can continue working locally, and send his data up later. If his personal computer isn't working, he can use another. Second, the response time provided by personal computers tends to be better than that of timeshared terminals linked to a mainframe. The user spends less time waiting, because the distance between the computer and the display is a few inches, and the bandwidth is higher. Third, by offloading processing that really can be done on the personal computer, response time is better on the main computer. One feature that would ensure the fastest possible response time for the user would be if the personal computer was always connected to the mainframe, and passed along those tasks for which mainframe computing power was required. There is no user productivity tool that automatically splits processing between the mainframe and micro, but they will certainly come along.

There are several input devices that can make a user productivity tool easier to use for novices, or those who don't have time to learn a proactive, apertural system. Touchscreens, light pens, and mice can all be used to select items from a menu. If someone pre-defines reports or activities for such a person, and builds a menu for them, they can use a powerful system without much effort at all. Touchscreens can be used on asynchronous terminals, 327X terminals, and personal computers. Light pens are supported on 327X devices and personal computers. Mice are available for personal computers. In each case, the user is made more productive by limiting his choices (and thus what he has to learn).

This section of the paper has explained a number of issues and choices that determine the character of a user productivity tool. To evaluate a specific product, you must analyse its character. Does it integrate or segregate data? Is it non-procedural? Is it reactive or proactive? Is it apertural or segmented? Does it process natural language input? Does it support special input/output devices? Give a weight to each issue discussed, and decide where the product falls. The weights should reflect the type of users and kinds of applications the product will be used for. No one product meets all the criteria, but if you carefully consider the design philosophy of the product, you will find the one that best meets your needs.

The first half of this paper suggested features that should be found in user productivity tools. This half explains methods for implementing those features in APL.

### **IMPLEMENTING INTEGRATED DATA**

Auxiliary processors provide the bridge between APL and external data. If a user productivity tool is written in APL, it must use AP's to import and export data. An auxiliary processor is simply a program independent of APL that provides services not available in the APL language itself.

APL programs communicate with AP's via shared variables. A shared variable is just a variable whose value is known to two programs simultaneously. System functions are provided in the APL language to manipulate shared variables. These functions can offer shares, control the sequence of access to a shared variable, query share status, and retract shares. Data is transmitted through the variable by simply assigning or using its value.

People who program in SHARP APL have several AP's to use in accessing data that is external to APL. The OS Dataset Auxiliary Processor (also known as TSIO) can be used to transfer data to and from a variety of sequential and direct file organizations. The VSAM Dataset Auxiliary Processor (VSDI) is used to transfer entry and key-sequenced data to and from VSAM files. The Function Call Auxiliary Processor (FCAP) allows APL programs to 'call' programs written in other languages. Such programs can be the batch interfaces to Data Base Management Systems.

There are many approaches to implementing integrated data. Each distinct source-target transfer can be special-cased. We will consider a more general approach instead.

Integrating data into the SHARP APL environment normally means putting it in APL files. But such transfers must be done through an APL workspace. So, we must first decide on how the data will be stored while it is in the workspace. Flat files are the simplest organization. Enclosed arrays are the simplest way to represent flat files. Enclosing data makes it possible to store different data types and array shapes in a single array. In the case of a flat file, the data is often treated as a matrix of enclosures. Rows correspond to records, and columns to fields. Numeric and character data can be stored in a single array, and even varying length fields are possible.

How exactly should enclosed arrays be structured? There is a large overhead for each enclosure. This makes enclosing every field/record combination use a lot of storage on file. It is, however, a very convenient representation. As long as the blocks of data read from the file aren't extremely large, the space temporarily consumed by lots of enclosures in the active work really doesn't matter. So, we will enclose each field/record combination within the workspace.

Using enclosed arrays on files is a different matter. If you want to do so, we strongly recommend that you enclose each record (as a vector) or each field (as a matrix). This brings down the storage overhead to a reasonable level. The functions *CELLSFROMRECORDS* and *CELLSFROMFIELDS*, shown in the function listing section, can convert the stored representation to the cell representation if needed.

Second, we must decide on how to get data into this intermediate form. The simplest approach is to require a set of cover functions for each file organization supported. There must be functions to open the file for reading and writing. These should return an identifier which will enable the read/write functions to send data to the proper place. There must be functions to read data from and write data to each type of file as well. The input function must handle blocking the data and putting it in an enclosed array. The output function must handle unblocking the data and putting it in the form the external file needs.

Third, we need a way to specify the parameters of a specific transfer request. These include the following.

- Input file name, type, and structure
- Output file name, type, and structure
- Record selection criteria
- Field selection criteria

The ability to extract just part of the source file is particularly important.

The transfer function needs the following inputs. The left argument is a description of the input file layout. It contains the name, start position, length, and data type of each field. The right argument contains the description of the transfer request. It specifies the file name, type, and structure of the input and output files. It also contains a character vector which is an expression that identifies the records to transfer, and a character matrix listing the fields to transfer.

The transfer function works as follows:

1. create a partition vector from the field starts and lengths
2. Map the field names into indices
3. Generate the following functions

*OPENΔSOURCE*

*OPENΔTARGET*

*GETΔSOURCE*

*PUTΔTARGET*

Use the code in the modules of the same names that correspond to the appropriate file types:

4. Generate the selection function
5. Open the files
6. Loop through the input file, reading a block of records, selecting those to be transferred, and writing them to the output file

The programs shown in the program listing section entitled "Integrated Data" implement this scheme. Support is provided for four file structures:

1. APL file containing character matrices,
2. APL file containing package variables
3. OS dataset, fixed unblocked
4. OS dataset, fixed blocked

It would take very few changes to the programs presented to support four additional file structures:

5. APL file containing numeric matrices
6. APL file containing vectors of enclosed arrays
7. VSAM entry sequenced datasets
8. VSAM key sequenced datasets

While the cover functions would be more complicated, the design could easily accommodate access to Data Base Management Systems.

## IMPLEMENTING PROACTIVE SYSTEMS

Proactive systems normally have a command language. Such languages range from very simple to quite complex. This section explains several methods for implementing such command languages.

### Keyword Matching

The most simple command language consists of operation/operand pairs. The name of the operation is followed by a blank and an operand, where appropriate. Such a command language is sufficient for situations such as scrolling. A typical language of this type is described in the diagram below. Items in brackets are optional. Text in small letters should be substituted for with a specific value. The | character indicates that one of the alternatives should be chosen.

```
UP      [number [ROWS] | [PAGES] ]
DOWN    [number [ROWS] | [PAGES] ]
LEFT    [number [COLS] | [PAGES] ]
RIGHT   [number [COLS] | [PAGES] ]
TOP
BOTTOM
```



With such a language, statements like the following would all be valid.

```
UP  
DOWN 10  
LEFT 3 COLS  
RIGHT 2 PAGES
```

The program shown in part 1 of the program listing section entitled 'Proactive Systems' implements this language. It checks for the operation name in the list of valid operations. If it is found, the program branches to the appropriate line to do the operation. It allows for unique, truncated abbreviations of operation names.

### **APL Syntax Analysis**

A more complicated command language allows for combining several keywords or parameters in a statement. One inexpensive approach to analyzing such languages for correct syntax is to let APL do it for you. This is possible if you structure your language to conform to APL syntax. Then all you have to do is define functions and variables that correspond to your keywords and parameters. When you execute the statement, if it doesn't create a syntax error, you have a valid statement in your language. If it does error out, you can catch it with an error trap, and report it to the user.

How do you know how to interpret the statement after it executes successfully? Each function concatenates a function code to its argument(s), and returns the result as an enclosed vector. The program which actually executes the expression loops through the final result, selecting values and executing the designated functions.

How do you prevent APL from executing a function represented by a symbol you wish to have another meaning? Just substitute the name of a defined function for the symbol before you execute it to check the syntax. This same technique also enables you to use APL symbols to mean the **same** thing in your language, while deferring processing.

An example of a more complicated command language is that required to support row selection by values in columns. The general form of the statement would be

WHERE (column relation value) logical (column relation value) logical ...

For each column in the data file, there is a variable with the same name. It contains the enclosure of the name of the column, i.e. *DATE* ← <'DATE'. The values must be either numbers or literals enclosed in quotes. The logical operations are AND, OR, XOR. The interesting part of this language is the support of relations. The following are provided.

APL Function	Command Language Keyword				
=    ∧.=	=	EQ	IS	EQUALS	IS EQUAL TO
≠    ∨.≠	<>	NE	ISNT	NOT EQUAL	ISNT EQUAL TO
>	>	GT	IS ABOVE	GREATER	IS GREATER THAN
<	<	LT	IS BELOW	LESS	IS LESS THAN
≥	>=	GE	ISNT BELOW		
≤	<=	LE	ISNT ABOVE		
∈			AMONG	IS AMONG	
<.<			BETWEEN	IS BETWEEN	
			ISNT AMONG		
			ISNT BETWEEN		

The programs shown in part 2 of the program listing section entitled 'Proactive Systems' implement this language. Several techniques are used to provide all of this flexibility. Certain keywords only serve to make the statement read more like English. The corresponding defined functions (*TO*, *THAN*) just return their argument if called monadically. If called dyadically, they signal an error.

Other keywords can have either a column name or another keyword on their left side. The corresponding defined functions (*EQUALS*, *EQUAL*, *ABOVE*, *BELOW*, *GREATER*, *LESS*, *AMONG*, *BETWEEN*) handle this by catenating an empty vector to their result. The function which follows decides what to do with this marker.

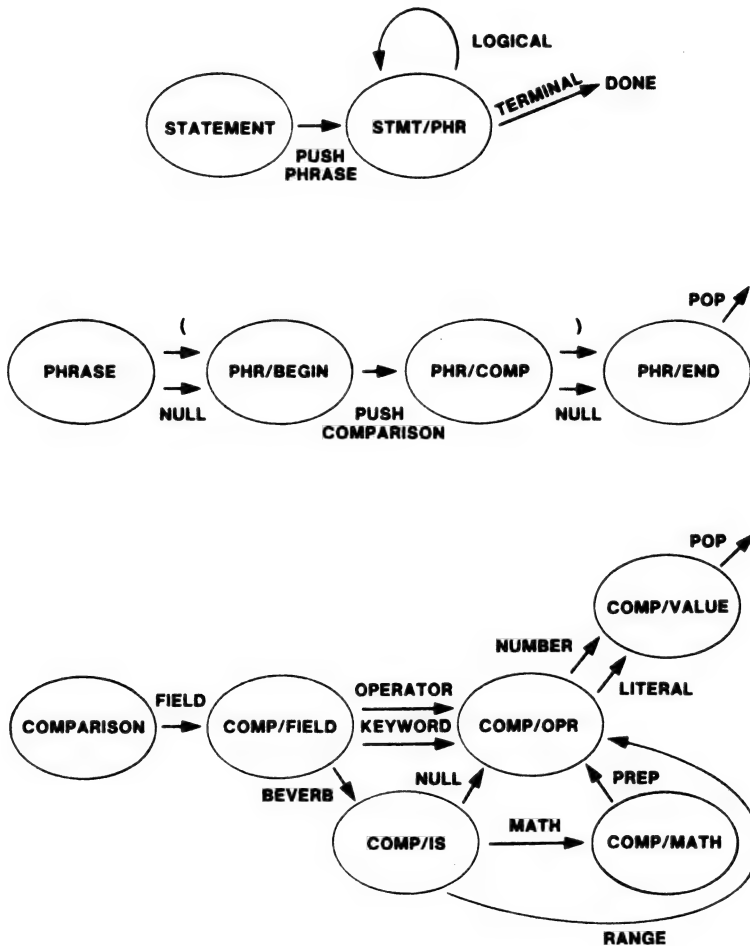
Finally, a couple of keywords serve both as syntactic fillers and as comparison functions. The corresponding defined functions (*IS*, *ISNT*) decide which they are by looking for a marker provided by a previous function.

### Augmented Transition Networks

The most sophisticated method of handling command languages is to use a formal grammar. One way to represent a formal grammar of any language (natural or artificial) is with a technique known as 'Augmented Transition Networks'. The possible valid configurations that a statement can take on are represented by network diagrams. The nodes are states, and the arcs between nodes represent valid transitions from one state to another. The networks are augmented with actions to take when a particular arc is traversed. Normally, more than one network is required to describe a language. This means that it is necessary to be able to put aside one network temporarily (push it on a stack) and return to it later.

The chief benefit of using the ATN formalism is complete generality. There are some constructs that can't be represented by an APL-like grammar. An example is a keyword that must be a function, and that may occur both in the middle and at the right end of a statement.

The diagrams that define the selection language described above are the following.



The programs to implement this grammar are found in part 3 of the the program listing section entitled 'Proactive Systems'. Basically, the main program tries each path from the current state, one at a time. If a path is taken, the cursor is advanced. If no path is taken, an error occurs. If a different network is selected, the current one is stacked for completion later.

## IMPLEMENTING APERTURAL SYSTEMS

The programmer who uses SHARP APL to implement apertural systems has an extremely powerful tool available. This is Auxiliary Processor 124, the full screen AP. This AP allows the programmer to divide the screen into logical fields, and to control the display and usage attributes of each field.

The programs in the section entitled "Apertural Systems" provide the framework for building a full screen application. They call subroutines provided for use with AP124 which are found in workspace 5 *IBM3270*.

The overall pattern of work with a screen is the following:

1. Initialize application
2. Initialize screen
3. While user hasn't said Quit
4. Reformat the screen, if necessary
5. Rewrite text in fields, if necessary
6. Set the cursor
7. Wait for user input
8. Process function keys, data cells, and commands
9. Recompute screen definition

This is the structure of the function *APERTURAL*.

## **CONCLUSION**

Software tools for users can really make them more productive if they provide integration of data and non-procedural processing. Interactive processing has changed quite a bit in character in the last few years. The old teletype/reactive systems provide less user productivity than the apertural/proactive systems of today. SHARP APL provides a very congenial environment for implementing such apertural/proactive systems. If it is the hub of your user application development, all the data and resources of the production data center can be placed in the hands of the user.

## **ACKNOWLEDGEMENTS**

My thanks to Dave Smith, Mike Powell, and Theo Sahlsberg of I.P. Sharp Associates for making a number of suggestions for improving this paper.

**EVALUATION CHECKLIST****Integrated Data**

- ☐ import from operating system flat files
- ☐ export to operating system flat files
- ☐ import from other user productivity tools
- ☐ export to other user productivity tools
- ☐ link records within user productivity tool

import from hierarchical data bases: \_\_\_\_\_

import from network data bases: \_\_\_\_\_

import from relational data bases: \_\_\_\_\_

- ☐ data dictionary accessible by users
- ☐ download/upload data to PC
- ☐ PC intelligent workstation
- ☐ direct access to data on other mainframes

**Non-procedural Processing**

☐ Reactive ☐ /Proactive

**Command language**

- ☐ consistent spelling
- ☐ consistent syntax
- ☐ consistent semantics

**Function keys**

- ☐ consistent usage
- ☐ protected from accidents
- ☐ keypad layout externally consistent

☐ Segmented ☐ Apertural

**User control of display**

- ☐ scroll horizontally and vertically

- \_\_\_\_\_ slice data horizontally and vertically
- \_\_\_\_\_ shuffle data (sort, move, hold)

Display design

- \_\_\_\_\_ program and data always identified
- \_\_\_\_\_ unnecessary details omitted
- \_\_\_\_\_ summary and detail screens
- \_\_\_\_\_ optimal use of screen attributes

Help facilities

- \_\_\_\_\_ checklist helps with carry-back
- \_\_\_\_\_ statement composition helps with carry-back
- \_\_\_\_\_ help by topic and screen area

I/O devices supported

- \_\_\_\_\_ graphics
- \_\_\_\_\_ personal computers
- \_\_\_\_\_ special data entry devices

**Integrated Data**

```

▽ COUNT←REQUEST TRANSFER LAYOUT;FLDNAMES;FLDSTARTS;FLDLENGTHS;
PARTN;INFILE;OUTFILE;RECSELECT;FLDSELECT;FLDINDEX;SOURCE;TARGET;
BLOCK
[1] A PURPOSE: TRANSFER SELECTED DATA FROM ONE FILE TO ANOTHER
[2] A RIGHT ARGUMENT: INPUT FILE LAYOUT(SEE BELOW)
[3] A LEFT ARGUMENT: TRANSFER REQUEST (SEE BELOW)
[4] A RESULT: NUMBER OF RECORDS TRANSFERRED
[5] FLDNAMES↔>LAYOUT[1] ◇ FLDSTARTS↔>LAYOUT[2] ◇ FLDLENGTHS
    ↔>LAYOUT[3]
[6] INFILE↔>REQUEST[1] ◇ OUTFILE↔>REQUEST[2]
[7] RECSELECT↔>REQUEST[3] ◇ FLDSELECT↔>REQUEST[4]
[8] A
[9] PARTN←((-1+FLDSTARTS)+-1+FLDLENGTHS)ρ0 ◇ PARTN[FLDSTARTS]←1
[10] FLDINDEX←FLDNAMES LOOKUP FLDSELECT
[11] (>OUTFILE[2]) BUILDFILEFNS>INFILE[2]
[12] FLDNAMES BUILD PICKREC RECSELECT
[13] A
[14] SOURCE←OPENSOURCE>INFILE[1] ◇ TARGET←OPENTARGET>OUTFILE[1]
[15] LOOP:BLOCK←GETSOURCE PARTN ◇ →(EOF=1)ρEND
[16] FLDNAMES[FLDINDEX;] PUTTARGET(PICKRECORDS BLOCK)[;FLDINDEX]
[17] →LOOP
[18] END: ▽

```

▽ R←X LOOKUP Y

```

[1] W←((-1+ρX)⌈-1+ρY ◇ X←((1+ρX),W)+X ◇ Y←((1+ρY),W)+Y
[2] R←⌈IO++/∧\Xv.≠QY ▽

```



```

▽ FIELDS BUILDPICKREC SELECT;CTR;LMT
[1] A PURPOSE: BUILDS RECORD SELECTION FUNCTION
[2] A RIGHT ARGUMENT: APL STATEMENT, USING FIELD NAMES
[3] A LEFT ARGUMENT: NAMES OF FIELDS IN INPUT FILE
[4] CTR←0 ◇ LMT←1⌈ρFIELDS
[5] LOOP:→(LMT<CTR←CTR+1)ρEND
[6] OLD←(FIELDS[CTR;]≠' ')/FIELDS[CTR;] ◇ NEW←'ω[;',(▼CTR),']'
[7] SELECT←SELECT STRINGREPLACE '/' ,OLD,'/' ,NEW
[8] →LOOP
[9] END:WIDTH←15⌈ρSELECT
[10] □FX(WIDTH↑'ω←PICKRECORDS ω'),[0.5] WIDTH↑SELECT ▽

```

```

▽ OUTTYPE BUILDFILEFNS INTYPE;FN
[1] A PURPOSE: BUILD FILE FNS FOR THIS TRANSFER
[2] A RIGHT ARGUMENT: 'AC', 'AP', 'FU', OR 'FB'
[3] A LEFT ARGUMENT: 'AC', 'AP', 'FU', OR 'FB'
[4] FN←□CR 'OPENSOURCEΔ',INTYPE
[5] FN[1;(FN[1;]⌈'Δ')+ 0 1 2]←' ' ◇ FN←□FX FN
[6] FN←□CR 'GETSOURCEΔ',INTYPE
[7] FN[1;(FN[1;]⌈'Δ')+ 0 1 2]←' ' ◇ FN←□FX FN
[8] FN←□CR 'OPENTARGETΔ',OUTTYPE
[9] FN[1;(FN[1;]⌈'Δ')+ 0 1 2]←' ' ◇ FN←□FX FN
[10] FN←□CR 'PUTTARGETΔ',OUTTYPE
[11] FN[1;(FN[1;]⌈'Δ')+ 0 1 2]←' ' ◇ FN←□FX FN ▽

```

```

▽ TIE←OPENSOURCEΔAC SOURCE;□TRAP;DATA
[1] A PURPOSE: OPEN APL CHARACTER FILE AS A SOURCE
[2] A ARGUMENT: CHAR VECTOR FILE NAME
[3] A GLOBALS SET: SCTR,SLMT,EOF
[4] A RESULT: SCALAR TIE NUMBER OR (0,ERROR CODE)
[5] □TRAP←'▽ 22 24 19 E →ERR'
[6] TIE←((120)∈□NUMS)10
[7] SOURCE □STIE TIE
[8] →(0≠-/2+□SIZE TIE)ρL1
[9] TIE← 0 4 ◇ →EXIT
[10] L1:DATA←□READ TIE,1+□SIZE TIE
[11] →(2≠ρρ□PNames DATA)ρL2
[12] TIE← 0 5 ◇ →EXIT
[13] L2:→(∼DATA⇒DATA)ρL3
[14] TIE← 0 5 ◇ →EXIT
[15] L3:→(0≠1+0ρDATA)ρOK
[16] TIE← 0 5 ◇ →EXIT
[17] OK:SCTR←-1+1+□SIZE SOURCE
[18] SLMT←-1+1+1+□SIZE SOURCE
[19] EOF←0 ◇ →EXIT
[20] ERR:TIE←0, 22 24 29 124ρ□ER
[21] EXIT: ▽

```

```

    ▽ TIE←OPENTARGET△AC TARGET;□TRAP;DATA
[1]  A PURPOSE: OPEN APL CHARACTER FILE AS A TARGET
[2]  A ARGUMENT: CHAR VECTOR FILE NAME
[3]  A RESULT: SCALAR TIE NUMBER OR (0,ERROR CODE)
[4]  □TRAP←'▽ 22 E →L0 ▽ 24 19 E →ERR'
[5]  TIE←((120)∈□NUMS)10
[6]  TARGET □STIE TIE
[7]  →L1
[8]  L0:TARGET □CREATE TIE
[9]  L1:→(0=-/2↑□SIZE TIE)ρEXIT
[10] L2:DATA←□READ TIE,1↑□SIZE TIE
[11] →(2≠ρ□PNames DATA)ρL3
[12] TIE← 0 5 ◇ →EXIT
[13] L3:→(∼DATA⇒DATA)ρL4
[14] TIE← 0 5 ◇ →EXIT
[15] L4:→(0≠1↑0ρDATA)ρEXIT
[16] TIE← 0 5
[17] ERR:TIE←0, 22 24 29 124ρ□ER
[18] EXIT: ▽

```

```

    ▽ BLOCK←GETSOURCE△AC CUT
[1]  A PURPOSE: GET BLOCK FROM APL CHARACTER FILE
[2]  A ARGUMENT: BOOLEAN VECTOR PARTITIONING DATA INTO FIELDS
[3]  A GLOBALS USED: SLMT, SCTR, EOF
[4]  A RESULT: MATRIX OF ENCLOSED ARRAYS
[5]  →(SLMT<SCTR←SCTR+1)ρENDFILE
[6]  BLOCK←□READ TIE,CTR
[7]  BLOCK←(((1↑ρBLOCK),+ /CUT)ρ((ρ,BLOCK)ρCUT) 1¨<BLOCK
[8]  →EXIT
[9]  ENDFILE:EOF←1 ◇ BLOCK← 0 0 ρ0
[10] EXIT: ▽

```

```

▽ NAMES PUTTARGET△AC BLOCK;□TRAP;□PS
[1]  A PURPOSE: PUT A BLOCK ON AN APL CHARACTER FILE
[2]  A RIGHT ARGUMENT: MATRIX OF ENCLOSED ARRAYS
[3]  A LEFT ARGUMENT: CHAR MATRIX OF FIELD NAMES
[4]  □TRAP←'▽ 21 E →FIX'
[5]  □PS← -1 -1 0 0
[6]  TRY:(▽BLOCK) □APPEND TARGET
[7]  →END
[8]  FIX:(⌈1.1×⌈/^-1+□SIZE TARGET) □RESIZE TARGET
[9]  →TRY
[10]END: ▽

```

```

▽ TIE←OPENSOURCE△AP SOURCE;□TRAP;DATA
[1]  A PURPOSE: OPEN APL PACKAGE FILE AS A SOURCE
[2]  A ARGUMENT: CHARACTER VECTOR FILE NAME
[3]  A GLOBALS SET: SCTR,SLMT,EOF
[4]  A RESULT: SCALAR TIE NUMBER OR (0,ERROR CODE)
[5]  □TRAP←'▽ 22 24 19 E →ERR'
[6]  TIE←((120)∈□NUMS)10
[7]  SOURCE □STIE TIE
[8]  →(0≠-/2+□SIZE TIE)ρL1
[9]  TIE← 0 4 ◇ →EXIT
[10] L1:DATA←□READ TIE,1+□SIZE TIE
[11] →(2=ρρ□PNAMES DATA)ρOK
[12] TIE← 0 5 ◇ →EXIT
[13] OK:SCTR←-1+1+□SIZE TIE
[14] SLMT←-1+1+1+□SIZE TIE
[15] EOF←0 ◇ →EXIT
[16] ERR:TIE←0, 22 24 29 124ρ□ER
[17]EXIT: ▽

```

```

    ▽ TIE←OPENTARGET△AP TARGET;□TRAP;DATA
[1] A PURPOSE: OPEN APL PACKAGE FILE AS TARGET
[2] A ARGUMENT: CHAR VECTOR FILE NAME
[3] A RESULT: SCALAR TIE NUMBER OR (0,ERROR CODE)
[4] □TRAP←'▽ 22 E →L0 ▽ 24 19 E →ERR'
[5] TIE←((120)∈□NUMS)10
[6] TARGET □STIE TIE
[7] →L1
[8] L0:TARGET □CREATE TIE
[9] L1:→(0=-/2↑□SIZE TIE)ρEXIT
[10] DATA←□READ TIE,1↑□SIZE TIE
[11] →(2=ρρ□PNAMES DATA)ρOK
[12] TIE← 0 5 ◇ →EXIT
[13]ERR:TIE←0, 22 24 29 144ρ□ER
[14]EXIT: ▽

```

```

    ▽ BLOCK←GETSOURCE△AP CUT;BLOCK;VARS;VCTR;VLMT;PACK
[1] A PURPOSE: GET BLOCK FROM APL PACKAGE FILE
[2] A ARGUMENT: BOOLEAN VECTOR PARTITIONING DATA INTO FIELDS
[3] A RESULT: MATRIX OF ENCLOSED ARRAYS
[4] →(SLMT<SCTR+SCTR+1)ρENDFILE
[5] PACK←□READ TIE,CTR
[6] VARS←□PNAMES PACK ◇ BLOCK←10
[7] VCTR←0 ◇ VLMT←1↑ρVARS
[8] LOOP:→(VLMT<VCTR+VCTR+1)ρEND
[9] BLOCK←BLOCK,<<1 VARS[VCTR] □PVAL PACK
[10] →LOOP
[11]END:BLOCK→BLOCK ◇ →EXIT
[12]ENDFILE:EOF←1 ◇ BLOCK← 0 0 ρ0
[13]EXIT: ▽

```

```

▽ NAMES PUTTARGET△AP BLOCK;□TRAP
[1] A PURPOSE: PUT BLOCK ON APL PACKAGE FILE
[2] A RIGHT ARGUMENT: MATRIX OF ENCLOSED ARRAYS
[3] A LEFT ARGUMENT: CHAR MATRIX OF FIELD NAMES
[4] □TRAP←'▽ 21 E →FIX'
[5] VCTR←0 ◇ VLMT←-1+ρBLOCK ◇ PACK←□PACK ''
[6] LOOP:→(VLMT<VCTR+VCTR+1)ρEND
[7] PACK←PACK □PINS NAMES[VCTR;] □PACK BLOCK[;VCTR]
[8] →LOOP
[9] END:PACK □APPEND TARGET
[10] →EXIT
[11]FIX:(⌈1.1×⌈/^-1+□SIZE TARGET) □RESIZE TARGET
[12] →TRY
[13]EXIT: ▽

```

```

▽ VAR←OPENSOURCE△FU SOURCE
[1] A PURPOSE: OPEN AN OS DATASET WITH FIXED UNBLOCKED RECORDS AS
  A SOURCE
[2] A ARGUMENT: CHAR VECTOR DATASET NAME
[3] A RESULT: CHAR VECTOR <CTL> VARIABLE OR (0,ERRORCODE)
[4] VAR←'CTL',(-8\ρSOURCE)+SOURCE
[5] -370 □SVO VAR
[6] L1:→(2=□SVO VAR)ρL2
[7] →□SCρL1
[8] L2:-1 □SVC VAR
[9] ⑆VAR,'←10'
[10] →(0≠1+⑆VAR)ρERR
[11] ⑆VAR,'←''SR DSN=',SOURCE,' RECFM=F'''
[12] →(0=1+⑆VAR)ρEND
[13]ERR:VAR← 0 1
[14]END:EOF←0 ▽

```

▽ VAR←OPENTARGET△FU SOURCE

```
[1] A PURPOSE: OPEN AN OS DATASET WITH FIXED UNBLOCKED RECORDS AS
    A TARGET
[2] A ARGUMENT: CHAR VECTOR DATASET NAME
[3] A RESULT: CHAR VECTOR <CTL> VAR, OR (0,ERRORCODE)
[4] VAR←'CTL',(-8|ρSOURCE)↑SOURCE
[5] -370 □SVO VAR
[6] L1:→(2=□SVO VAR)ρL2
[7] →□SCρL1
[8] L2:-1 □SVC VAR
[9] 1VAR,'←10'
[10] →(0≠1↑1VAR)ρERR
[11] 1VAR,'←1'SR DSN=',SOURCE,' RECFM=F DISP=',WRITE, ''
[12] →(0=1↑1VAR)ρEND
[13]ERR:VAR← 0 1
[14]END: ▽
```

▽ BLOCK←GETSOURCE△FU CUT

```
[1] A PURPOSE: GET BLOCK FROM OS DATASET, FIXED UNBLOCKED
[2] A ARGUMENT: BOOLEAN VECTOR IDENTIFYING FIELDS
[3] A RESULT: MATRIX OF ENCLOSED ARRAYS
[4] BLOCK←(0,+/CUT)ρ0
[5] LOOP:RECORD←1SOURCE
[6] →(~0∈ρRECORD)ρDATA
[7] RECORD←1SOURCE
[8] →(0≠1↑,RECORD)ρDATA
[9] EOF←1 ◇ →END
[10]DATA:BLOCK←BLOCK,[□IO] CUT 1◌◌<RECORD
[11] →(10000>4 □WS 'BLOCK')ρLOOP
[12]END: ▽
```

```

▽ NAMES PUTTARGET△FU BLOCK;□PS;RCTR;RLMT
[1] A PURPOSE: PUT BLOCK ON OS DATASET, FIXED UNBLOCKED
[2] A RIGHT ARGUMENT: MATRIX OF ENCLOSED ARRAYS
[3] A LEFT ARGUMENT: CHAR MATRIX OF FIELD NAMES
[4] □PS← -1 -1 0 0
[5] RCTR←0 ◇ RLMT←ρBLOCK
[6] L3:→(RLMT<RCTR<RCTR+1)ρL4
[7] ⊕TARGET, '←▽BLOCK[RCTR;]'
[8] →(0=ERR←⊕TARGET)ρL3
[9] □←'WRITE ERROR'
[10]L4: ▽

```

```

▽ VAR←OPENSOURCE△FB SOURCE
[1] A PURPOSE: OPEN AN OS DATASET WITH FIXED BLOCKED RECORDS AS A
    SOURCE
[2] A ARGUMENT: CHAR VECTOR DATASET NAME
[3] A RESULT: CHAR VECTOR <CTL> VAR NAME, OR (0,ERRORCODE)
[4] VAR←'CTL', (-8└ρSOURCE)↑SOURCE
[5] -370 □SVO VAR
[6] L1:→(2=□SVO VAR)ρL2
[7] →□SCρL1
[8] L2:-1 □SVC VAR
[9] ⊕VAR, '←└0'
[10] →(0≠1↑⊕VAR)ρERR
[11] ⊕VAR, '←' 'SR DSN=', SOURCE, ' RECFM=FBS' ''
[12] →(0=1↑⊕VAR)ρEND
[13]ERR:VAR← 0 1
[14]END:EOF←0 ▽

```



```

    ▽ VAR←OPENTARGETΔFB SOURCE
[1]  ρ PURPOSE: OPEN OS DATASET WITH FIXED BLOCKED RECORDS AS TARGET
[2]  ρ ARGUMENT: CHAR VECTOR DATASET NAME
[3]  ρ RESULT: CHAR VECTOR <CTL> VARIABLE OR (0,ERRORCODE)
[4]  VAR←'CTL',(-8\ρSOURCE)↑SOURCE
[5]  -370 □SVO VAR
[6]  L1:→(2=□SVO VAR)ρL2
[7]  →□SCρL1
[8]  L2:-1 □SVC VAR
[9]  $VAR,'←10'
[10] →(0≠1↑$VAR)ρERR
[11] $VAR,'←'SW DSN=',SOURCE,' RECFM=FBS DISP=',WRITE, ''
[12] →(0=1↑$VAR)ρEND
[13] ERR:VAR← 0 1
[14] END: ▽

```

```

    ▽ BLOCK←GETSOURCEΔFB SOURCE
[1]  ρ PURPOSE: GET BLOCK FROM OS DATASET, FIXED BLOCKED
[2]  ρ ARGUMENT: CHAR VECTOR <CTL> VAR
[3]  ρ RESULT: MATRIX OF ENCLOSED ARRAYS
[4]  BLOCK←$SOURCE
[5]  →(~0∈ρBLOCK)ρL1
[6]  BLOCK←$SOURCE
[7]  →(0≠1↑,BLOCK)ρL1
[8]  EOF←1 ◇ →EXIT
[9]  L1:BLOCK←(((1↑ρBLOCK),+/CUT)ρ((ρ,BLOCK)ρCUT) 1◡<BLOCK
[10] EXIT: ▽

```

```

    ▽ NAMES PUTTARGETΔFB BLOCK;□PS
[1]  ρ PURPOSE: PUT BLOCK ON OS DATASET, FIXED BLOCKED
[2]  ρ RIGHT ARGUMENT: MATRIX OF ENCLOSED ARRAYS
[3]  ρ LEFT ARGUMENT: CHAR MATRIX OF FIELD NAMES
[4]  □PS← -1 -1 0 0
[5]  $TARGET,'←▽BLOCK'
[6]  →(0=ERR←$TARGET)ρEND
[7]  □←'WRITE ERROR'
[8]  END: ▽

```

## Proactive Systems

## Part 1

```

▽ ERROR←SCROLLΔLANG CMND;POSN;PARM;LIST;VALUE;QUAL;INDEX
[1] A PROCESSES THE FOLLOWING COMMANDS:
[2] A   UP      [N [ROWS] | [PAGES] ]
[3] A   DOWN   [N [ROWS] | [PAGES] ]
[4] A   RIGHT  [N [COLS] | [PAGES] ]
[5] A   LEFT   [N [COLS] | [PAGES] ]
[6] A   TOP
[7] A   BOTTOM
[8] A ARGUMENT- CHAR VECTOR SCROLLING LANGUAGE STATEMENT
[9] A GLOBAL VARS: CURSOR, PAGESIZE, NUMROWS, NUMCOLS
[10] RESULT: ERROR FLAG (0 OR 1)
[11] LIST← 6 6 ρ'UP DOWN RIGHT LEFT TOP BOTTOM' ◇ ERROR←0
[12] CMND←(v\ ' '≠CMND)/CMND
[13] POSN←CMNDι ' ' ◇ PARM←POSN+CMND ◇ CMND←(POSN-1)+CMND
[14] INDEX←(((1+ρLIST),ρCMND)+LIST)∧.=CMND)/ι1+ρLIST
[15] →(1=ρINDEX)ρL0 ◇ ERROR←1 ◇ →EXIT
[16] L0:POSN←PARMι ' ' ◇ QUAL←'P'=POSN+PARM ◇ VALUE←1⌈1+□FI(POSN-
1)+PARM
[17] →(L1,L2,L3,L4,L5,L6)[INDEX]
[18] L1:CURSOR[1]←CURSOR[1]-VALUE×PAGESIZE[1]*QUAL ◇ →EXIT
[19] L2:CURSOR[1]←CURSOR[1]+VALUE×PAGESIZE[1]*QUAL ◇ →EXIT
[20] L3:CURSOR[2]←CURSOR[2]-VALUE×PAGESIZE[2]*QUAL ◇ →EXIT
[21] L4:CURSOR[2]←CURSOR[2]+VALUE×PAGESIZE[2]*QUAL ◇ →EXIT
[22] L5:CURSOR← 1 1 ◇ →EXIT
[23] L6:CURSOR←NUMROWS,NUMCOLS ◇ →EXIT
[24] EXIT: ▽

```

**Part 2**

```

    ∇ R←SLICEΔLANG STMT;□TRAP
[1] A PURPOSE: ANALYZE SLICE LANGUAGE STATEMENT
[2] A ARGUMENT- CHAR VECTOR STATEMENT
[3] A RESULT- EMPTY VECTOR (IF ERROR) OR ENCLOSED VECTOR PARSE
[4] STMT←TRANSFORM STMT
[5] □TRAP←'∇ 2 6 11 E →L1'
[6] R←1STMT ◇ →L2
[7] L1:R←10
[8] L2: ∇

```



```

[7]  OLD ← ' = ' > ' < ' > ' > ' < ' > ' > ' ε ' > ' < . < '
[8]  ΔY ← ΔY
[9]  → (0 = ρ > ' ' ρ ΔY) ρ L1
[10] ΔZ ← ΔX > ' ≠ ' ΔY
[11] → L2
[12] L1: ΔY ← 1 + ΔY
[13] ΔY[□IO] ← NEW[OLD, ΔY[□IO]]
[14] ΔZ ← ΔX > ΔY
[15] L2: ∇

```

$\nabla \Delta Z \leftrightarrow TO \Delta X$   
 [1]  $\Delta Z \leftrightarrow \Delta X$   
 [2]  $\mu < THAN > IS IDENTICAL \nabla$

```

      ▽ ΔZ←ΔX EQUAL ΔY
[1]  1(0=□NC 'ΔX')/'ΔX←10'
[2]  ΔZ←ΔX▷='▷ΔY
[3]  A CHANGE = TO < > < > ∈ <.< FOR
[4]  A      <BELOW><ABOVE><LESS><GREATER><AMONG><BETWEEN> ▽

```

## Part 3

```

▽ PARSE←SLICEΔLANGΔATN EXPRESSION;CURRENT;CURSOR;INCLASS
[1] A PURPOSE: PARSE SLICE LANGUAGE STATEMENT USING ATN
[2] A ARGUMENT: TOKENIZED SLICE LANGUAGE EXPRESSION
[3] A RESULT: EMPTY VECTOR (FOR INVALID EXPRESSIONS) OR TOKEN CLASS
    LIST
[4] INITΔATN ◇ EXPRESSION←PREPARE EXPRESSION ◇ STACK←10 ◇ PARSE←10
[5] CURRENT←STATEMENT ◇ CURSOR←1 ◇ INCLASS←CLASSIFY EXPRESSION
    [CURSOR]
[6] A      NETWORK 1
[7] STATEMENT:→PUSH PHRASE,STMTPHR
[8] STMTPHR:→PATH STMTPHR,LOGICAL,STATEMENT
[9] →PATH STMTPHR,TERMINAL,EXIT
[10] →ERROR
[11]A      NETWORK 2
[12]PHRASE:→PATH PHRASE,RPAREN,PHRBEGIN
[13] →JUMP PHRBEGIN
[14]PHRBEGIN:→PUSH COMPARE,PHRCOMP
[15]PHRCOMP:→PATH PHRCOMP,LPAREN,PHREND
[16] →JUMP PHREND
[17]PHREND:→POP
[18]A      NETWORK 3
[19]COMPARE:→PATH COMPARE,FIELD,COMPFIELD
[20] →ERROR
[21]COMPFIELD:→PATH COMPFIELD,OPERATOR,COMPOPR
[22] →PATH COMPFIELD,KEYWORD,COMPOPR
[23] →PATH COMPFIELD,BEVERB,COMPIS
[24] →ERROR
[25]COMPOPR:→PATH COMPOPR,NUMBER,COMPVALUE
[26] →PATH COMPOPR,LITERAL,COMPVALUE
[27] →ERROR
[28]COMPIS:→PATH COMPIS,RANGE,COMPOPR
[29] →PATH COMPIS,MATH,COMPMATH
[30] →JUMP COMPOPR
[31]COMPMATH:→PATH COMPMATH,PREP,COMPOPR
[32] →ERROR
[33]COMPVALUE:→POP
[34]A
[35]ERROR:CURSOR SHOWERR EXPRESSION ◇ PARSE←10
[36]EXIT:CLEANUPΔFSM ▽

```

▽ INITΔATN

- [1] A PURPOSE: INITIALIZE TOKEN CLASSES
- [2] FIELD←1 ◇ NUMBER←2 ◇ LITERAL←3
- [3] OPERATOR←4 ◇ KEYWORD←5 ◇ BEVERB←6 ◇ MATH←7 ◇ RANGE←8 ◇ PREP←9
- [4] RPAREN←10 ◇ LPAREN←11 ◇ LOGICAL←12 ◇ TERMINAL←13 ◇ OTHER←14 ▽

▽ EXPR←PREPARE EXPR

- [1] A PURPOSE: PREPARE STATEMENT FOR PARSING WITH ATN
- [2] A ARGUMENT: CHAR VECTOR CONTAINING 'SLICE' LANGUAGE STATEMENT
- [3] A RESULT: ENCLOSED ARRAY OF TOKENS, OPERATORS CONVERTED TO KEYWORDS
- [4] EXPAND←EXPR∈'()'<>=' ◇ EXPAND←EXPAND^~1+EXPAND,0
- [5] EXPR←((,EXPAND,[1.1] 1)/,EXPAND,[1.1]~EXPAND)\EXPR
- [6] EXPAND←EXPR∈'()'<>=' ◇ EXPAND←EXPAND^~1+0,EXPAND
- [7] EXPR←((,1,[1.1] EXPAND)/,(~EXPAND),[1.1] EXPAND)\EXPR
- [8] EXPR←((EXPR≠' ')∨EXPR≠1φEXPR)/EXPR
- [9] EXPR←((' '≠1+EXPR)ρ' '),(-1×' '=1+EXPR)↓EXPR
- [10] EXPR←(-1◊<EXPR),(,';')>' ' ▽

▽ TYPE←CLASSIFY TOKEN;TEXT;OPRS;KEYWS;RANGES;MATHS

- [1] A PURPOSE: CLASSIFY TOKEN TYPE
- [2] A ARGUMENT: ENCLOSED CHAR VECTOR
- [3] A RESULT: INTEGER TYPE, AS DEFINED IN <INITΔATN>
- [4] OPRS←(, '='>(, '<'>(, '>'>'<='>'>='>'<>'
- [5] KEYWS←'EQ'>'NE'>'GT'>'LT'>'GE'>'LE'
- [6] RANGES←'ABOVE'>'BELOW'>'BETWEEN'>'AMONG'
- [7] MATHS←'EQUAL'>'LESS'>'GREATER'
- [8] TEXT←>TOKEN
- [9] →(∼∨/TOKEN≡◊>FIELDS)ρL1 ◇ TYPE←FIELD ◇ →EXIT
- [10] L1:→(∼□VI TEXT)ρL2 ◇ TYPE←NUMBER ◇ →EXIT
- [11] L2:→(∼('' '=1+TEXT)^'' '=1+TEXT)ρL3 ◇ TYPE←LITERAL ◇ →EXIT
- [12] L3:→(∼∨/TOKEN≡◊>OPRS)ρL4 ◇ TYPE←OPERATOR ◇ →EXIT
- [13] L4:→(∼∨/TOKEN≡◊>KEYWS)ρL5 ◇ TYPE←KEYWORD ◇ →EXIT
- [14] L5:→(∼∨/TOKEN≡◊>'IS'>'ISNT')ρL6 ◇ TYPE←BEVERB ◇ →EXIT
- [15] L6:→(∼∨/TOKEN≡◊>RANGES)ρL7 ◇ TYPE←RANGE ◇ →EXIT
- [16] L7:→(∼∨/TOKEN≡◊>MATHS)ρL8 ◇ TYPE←MATH ◇ →EXIT
- [17] L8:→(∼∨/TOKEN≡◊>'TO'>'THAN')ρL9 ◇ TYPE←PREP ◇ →EXIT
- [18] L9:→(∼∨/TOKEN≡◊>'AND'>'OR')ρL10 ◇ TYPE←LOGICAL ◇ →EXIT
- [19] L10:TYPE←(RPAREN,LPAREN,TERMINAL,OTHER)['(',')';1+TEXT]
- [20] EXIT: ▽

▽ TARGET←ACTION PATH ARG

```
[1] A PURPOSE: ATTEMPT TO TRAVERSE A PATH IN AN ATN
[2] A RIGHT ARGUMENT: CURRENT NODE,REQUIRED CLASS,TARGET
[3] A LEFT ARGUMENT: CHAR VECTOR ACTION TO EXECUTE IF TRAVERSED
[4] A RESULT: NEXT LINE TO BRANCH TO
[5] TARGET←10
[6] →(ARG[1 2]∨.≠CURRENT,INCLASS)ρEXIT
[7] ⚡(0≠[NC 'ACTION'])/'ACTION'
[8] PARSE←PARSE,ARG[2]
[9] TARGET←CURRENT←ARG[3]
[10] CURSOR←CURSOR+1
[11] INCLASS←CLASSIFY EXPRESSION[CURSOR]
[12]EXIT: ▽
```

▽ TARGET←JUMP TARGET

```
[1] A ARGUMENT/RESULT: NEXT LINE TO BRANCH TO
[2] CURRENT←TARGET ▽
```

▽ TARGET←PUSH LABELS

```
[1] A PURPOSE: CALL NETWORK AS SUBROUTINE
[2] A ARGUMENT: SUBNETWORK LOCATION, RETURN ADDRESS
[3] A RESULT: NEXT LINE TO BRANCH TO
[4] STACK←(1↑LABELS),STACK
[5] TARGET←CURRENT←1↑LABELS
[6] PARSE←PARSE,0 ▽
```

▽ TARGET←POP

```
[1] A PURPOSE: RETURN FROM NETWORK CALLED AS SUBROUTINE
[2] A RESULT: NEXT LINE TO BRANCH TO
[3] TARGET←CURRENT←1↑STACK ◇ STACK←1↑STACK
[4] PARSE←PARSE,-1 ▽
```

▽ CURSOR SHOWERR STATEMENT;LENGTH;POINTER;[PS

```
[1] A PURPOSE: SHOW LOCATION OF SYNTAX ERROR
[2] [PS←-1-1 0 1 ◇ [←'SYNTAX ERROR' ◇ [←STATEMENT
[3] LENGTH←1+,ρ>STATEMENT ◇ POINTER←(+/LENGTH)ρ' '
[4] POINTER[-1++/CURSOR+LENGTH]←'↑' ◇ [←POINTER ▽
```



**Apertural Systems**

```

▽ APERTURAL;FMT;TXT;CHGCODE;CHGPOSN;CHGFLDS;CHGTXTS
[1]  ρ SKELETON OF APERTURAL SYSTEM
[2]  INITΔAP124 ◇ INITΔAPPLICATION ◇ DEFINEΔSCREEN
[3]  LOOP:→(STOPFLAG=1)ρEND
[4]  →(FMTCHGFLAG=0)ρL1 ◇ FORMAT SCRNFMT
[5]  L1:→(VALCHGFLAG=0)ρL2 ◇ CHGFLDS WRITE SCRNTXT[CHGFLDS;]
[6]  L2:CURFLD SETCURSOR CURROW,CURCOL
[7]  SPLITENTRY READSCREEN ◇ CHGTXTS←READDATA
[8]  DOΔFNKEYS ◇ DOΔCELLS ◇ DOΔCOMMANDS
[9]  DEFINEΔSCREEN ◇ →LOOP
[10] END: ▽

```

```

▽ INITΔAP124;□IO
[1]  ρ INITIALIZE CONSTANTS USED WITH FULLSCREEN AP
[2]  CHARIN←0 ◇ NUMIN←1 ◇ CHAROUT←2
[3]  BLANK←0 ◇ REGULAR←1 ◇ HIGH←2
[4]  BLUE← 1 1 ◇ RED← 1 2 ◇ PINK← 1 3 ◇ GREEN← 1 4
[5]  TURQ← 1 5 ◇ YELLOW← 1 6 ◇ WHITE← 1 7
[6]  SYMBSET←0
[7]  NORMAL←0 ◇ BLINK←1 ◇ REVERSE←2 ◇ UNDERSCORE←3
[8]  SKIP←0 ◇ NOSKIP←1
[9]  DEFAULT←SYMBSET,NORMAL,SKIP,3
[10] □IO←1 ◇ VBAR←□AV[242] ◇ LTEE←□AV[243] ◇ RTEE←□AV[244]
[11] UTEE←□AV[245] ◇ DTEE←□AV[246] ◇ HBAR←□AV[247] ◇ CROSS←□AV[248]
[12] LRIGHT←□AV[249] ◇ LLEFT←□AV[250] ◇ ULEFT←□AV[251] ◇ URIGHT←□AV[252]
[13] SCRNSIZE← 24 80 ◇ STOPFLAG←0 ◇ FMTCHGFLAG←VALCHGFLAG←1
[14] CURROW←CURCOL←1 ◇ CURFLD←2
[15] →SHARE124 ▽

```

```

▽ INITΔAPPLICATION
[1]  P INITIALIZE VARIABLES DISPLAYED BY APPLICATION
[2]  PROGID←20↑'YOUR APPLICATION'
[3]  DATAID←22↑'YOUR FILE'
[4]  COMMAND←''
[5]  MESSAGE←''
[6]  POSITION←1,(1↑COUNTCELLS),1,~1↑COUNTCELLS
[7]  HEADINGS←GETHEADINGS\0
[8]  WIDTH←(,ρö>HEADINGS)[(,ρö>1 GETCELLS\0)
[9]  COLINDEX←CURCOL+~1+~1+(80>+\WIDTH)\0
[10] CELLS←ROWINDEX GETCELLS COLINDEX
[11] PFKEYS←'PF 1=HELP 3=STOP 4=INS 5=DLT 7=UP 8=DOWN 10=LEFT 11=
      RIGHT' ▽

```

```

∇ DEFINEΔSCREEN;FMT;TXT
[1]  A ASSIGN FMT AND TXT VARIABLES
[2]  FMT← 0 11 ρ0
[3]  TXT← 0 0 ρ''
[4]  A PROGRAM IDENTIFICATION
[5]  FMT←FMT,[1] 1 1 , 1 28 ,CHAROUT,BLUE,DEFAULT
[6]  TXT←TXT JOIN 28ρHBAR
[7]  FMT←FMT,[1] 1 30 , 1 20 ,CHARIN,TURQUOISE,DEFAULT
[8]  TXT←TXT JOIN 20↑PROGID
[9]  FMT←FMT,[1] 1 51 , 1 28 ,CHAROUT,BLUE,DEFAULT
[10] TXT←TXT JOIN 28ρHBAR
[11] A COMMAND AREA
[12] FMT←FMT,[1] 2 1 1 79 ,CHARIN,YELLOW,DEFAULT
[13] TXT←TXT JOIN 79↑COMMAND
[14] COMMANDFIELD←1↑ρFMT
[15] A ERROR/MESSAGE AREA
[16] FMT←FMT,[1] 3 1 1 79 ,CHAROUT,WHITE,0,BLINK,SKIP,0
[17] TXT←TXT JOIN 79↑MESSAGE
[18] A HOME POSITION
[19] FMT←FMT,[1] 4 1 , 1 3 ,CHAROUT,BLUE,DEFAULT
[20] TXT←TXT JOIN 'ROW'
[21] FMT←FMT,[1] 4 5 , 1 3 ,NUMIN,TURQUOISE,DEFAULT
[22] TXT←TXT JOIN 3 0 ▽POSITION[1]
[23] FMT←FMT,[1] 4 9 , 1 14 ,CHAROUT,BLUE,DEFAULT
[24] TXT←TXT JOIN 'OF ',(3 0 ▽POSITION[2]),', COLUMN'
[25] FMT←FMT,[1] 4 24 , 1 3 ,NUMIN,TURQUOISE,DEFAULT
[26] TXT←TXT JOIN 3 0 ▽POSITION[3]
[27] FMT←FMT,[1] 4 28 , 1 7 ,CHAROUT,BLUE,DEFAULT
[28] TXT←TXT JOIN ' OF ', 3 0 ▽POSITION[4]
[29] A DATA IDENTIFICATION
[30] FMT←FMT,[1] 5 1 , 1 27 ,CHAROUT,BLUE,DEFAULT
[31] TXT←TXT JOIN 27ρHBAR
[32] FMT←FMT,[1] 5 29 , 1 22 ,CHARIN,TURQUOISE,DEFAULT
[33] TXT←TXT JOIN 22↑DATAID
[34] FMT←FMT,[1] 5 52 , 1 27 ,CHAROUT,BLUE,DEFAULT
[35] TXT←TXT JOIN 27ρHBAR
[36] A FUNCTION KEYS
[37] FMT←FMT,[1] SCRNSIZE[1],1,1,(SCRNSIZE[2]-1),CHAROUT,BLUE,DEFAULT
[38] WIDTH←0.5×SCRNSIZE[2]-3+ρPFKEYS
[39] TXT←TXT JOIN((\WIDTH)ρHBAR),' ',PFKEYS,' ',(\WIDTH)ρHBAR
[40] A COLUMN HEADINGS
[41] FMT←FMT,[1] 6 1 , 1 79 ,CHAROUT,RED,DEFAULT
[42] TXT←TXT JOIN,▽HEADINGS

```

```

[43] DATACELLFIELDS←1↑ρFMT
[44]A      DATA CELLS
[45] FMT←FMT,[1] 7 1 ,(SCRNSIZE[1]-7),79,CHARIN,GREEN,DEFAULT
[46] TXT←TXT JOIN,▼CELLS
[47] DATACELLFIELDS←DATACELLFIELDS↓11↑ρFMT
[48]A
[49] SCRNFMT←FMT ◇ SCRNTXT←TXT ◇ CHGFLDS←11↑ρSCRNFMT ▽

```

▽ SPLITENTRY ENTRY

```

[1] CHGCODE←ENTRY[1 2] ◇ CHGPOSN←ENTRY[3 4 5]
[2] CHGFLDS←5↓ENTRY ◇ CURFLD←1↑CHGFLDS,1 ▽

```

## ▽ DO△FNKEYS

```

[1] →(CHGCODE[1]≠1)ρPAKEYS
[2] →(CHGCODE[2]=12)/L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12
[3] L1:HELP CURFLD ◇ →EXIT
[4] L2:
[5] →EXIT
[6] L3:STOPFLAG←1 ◇ →EXIT
[7] L4:
[8] →EXIT
[9] L5:
[10] →EXIT
[11] L6:
[12] →EXIT
[13] L7:CURROW←CURROW-1 ◇ CURROW←(1*CURROW<1)+CURROW,SCRNSIZE[1]
[14] →EXIT
[15] L8:CURROW←CURROW+1 ◇ CURROW←(1*CURROW>SCRNSIZE)+CURROW,1
[16] →EXIT
[17] L9:
[18] →EXIT
[19] L10:CURCOL←CURCOL-1 ◇ CURCOL←(1*CURCOL<1)+CURCOL,SCRNSIZE[2]
[20] →EXIT
[21] L11:CURCOL←CURCOL+1 ◇ CURCOL←(1*CURCOL>SCRNSIZE[2])+CURCOL,1
[22] →EXIT
[23] L12:
[24] →EXIT
[25] PAKEYS:→(CHGCODE[1]≠4)ρEXIT
[26] →(CHGCODE[2]=12)/L13,L14
[27] LPA1:
[28] →EXIT
[29] LPA2:
[30] →EXIT
[31] EXIT: ▽

```

## ▽ DO△CELLS

```

[1] →(∼v/DATACELLFIELDS∈CHGFIELDS)ρEND
[2] END: ▽

```

## ▽ DO△COMMANDS

```

[1] →(∼COMMANDFIELD∈CHGFIELDS)ρEND
[2] END: ▽

```



# **Managing the Information Revolution**

Session leader: David Keith, I. P. Sharp Associates

Speakers: James L. Schmit, Director of Research & Development, Computer Assisted Analysis  
Keith W. Iverson, Morgan Stanley & Company, Inc.  
Gösta Olavi, Senior Systems Analyst, Skandinaviska Enskilda Banken  
Eric B. Herr, Vice-President, Planning and Development, McGraw-Hill

## **Abstract**

Information flow to and from end users is another area directly affected by changing technologies. In this information revolution it is important that we match these new technologies with end-user requirements. Increasing amounts of data must be distributed to information consumers in a variety of new ways. Your users' needs may be met simply with a broadcast feed to a desktop micro for systems such as stock quotations or news. Or users may demand two-way communication with a central data base, in applications such as trading systems and electronic mail. Others may require a combination of both. This session will examine the information revolution by investigating a variety of recently implemented information systems on a case-by-case basis. Insight gained from this session should help any organization in responding to users' needs and planning information systems.





Selection of the word 'revolution' in the title for this session was not an easy choice. Other words that came to mind include 'evolution', 'explosion', and 'glut'. All of these words were reasonable candidates. How appropriate is 'revolution'? Webster's defines revolution as: "a sudden, radical, or complete change." The fact is that here in 1984, we are on the verge of a phenomenal change in the way that information is distributed, and the ways in which users interact with information. The change is not so much a change that users of information are demanding as it is a sudden change in the technology by which information can be conveyed. The driving force behind this transformation is the advent of the microcomputer. It is difficult to imagine that any development in the computing profession in the next 20 years will result in as rapid a change in the status quo.

To understand this revolution a little better, let us look at the changes in technology over the past 20 years. We can classify the years 1965 to 1980 as the pre-micro years. In the early part of that timeframe, if you wanted information that was computer-generated you most likely got it as a highspeed print, and probably lots of it too. True, the computer could sort the output in various ways, but you could still spend a good amount of time scanning the reams of output for the piece or pieces of information you were after. The timeliness of data was not quite as important then—if you got last month's revenue figures before the end of the following month, that was OK, and if you got yesterday's stock market prices today, that was OK too.

But during the late 1960's computer terminals (now known as 'dumb' terminals) became available, and software experts wrote software that permitted users to make requests interactively for information from a remote terminal to a centralized computer. The APL\360 language from IBM is a classic example of this development. Subsequent enhancements to APL\360, such as the addition of a file subsystem, permitted the data base people to build relatively large and useful data bases. When a user wanted something, a few simple commands permitted its retrieval and display. Many companies took advantage of this new capability and built private information systems that must have been successful, because management kept on asking for more, much to the delight of IBM. Still other companies jumped on the bandwagon by building 'public' data bases. Each year, the number of new public data bases available to anyone who is willing to pay for them increases by approximately 30%.

I. P. Sharp Associates has been active in this area since the early 1970's. We now have over 120 separate data bases (some of them with many files), occupying a total of approximately 50 billion bytes, making us one of the largest holders of public decision-making data in the world. In a sense, we are a typical example of how the information-gathering and distribution process worked in the pre-micro era, as illustrated in the following diagram.

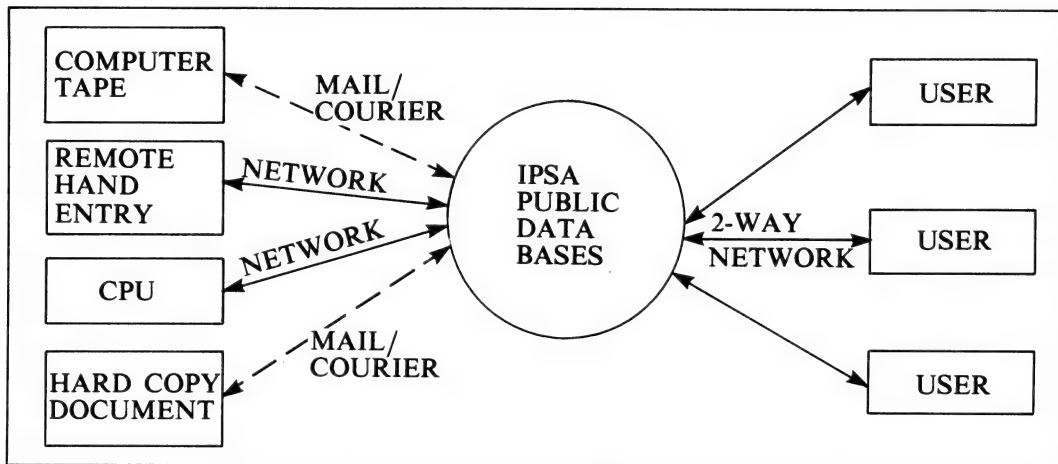


Figure 1

In this simplified scenario, data is sent only in one direction to the centralized data base, and is used to update the existing data. For example, the I. P. Sharp USSTOCK data base contains 10 years of daily history for over 10,000 stocks traded on U.S. stock exchanges. Each day, we receive via our network a 1 megabyte transmission of the most recent trading activity for each active stock, and use that data to update the master data base. Once that is done, users on 'dumb' terminals can access the data they want—perhaps the statistics for only 10 stocks of interest. They may want the most recent day, or they may want summarized weekly history for the past two years. In any event, it is important to note that they need a two-way network in order to be able to make such a request, and they are quite likely to expend CPU on the mainframe computer if they want to do anything with that data, such as plot it or perform a moving average on it.

Before leaving this era, let us look at another pre-micro method of data distribution as exemplified by the likes of Reuters and Quotron. Early on, these companies appreciated the high value of timely data, and they have been successful at turning this realization into highly profitable business ventures. They specialize in being able to deliver relatively small amounts of information to a user's terminal at what appears to be a very high rate. The information is usually page-oriented. For example, foreign exchange dealers might be interested in knowing what rates of exchange a large bank is currently posting. They must know what these rates are within seconds of their posting. This information is not voluminous—it can usually fit on a single 23 by 80 character page.

To avoid having to transmit the entire page from a central source to a user every time a user happens to ask for it, a copy of the formatted page is sent to all users hooked onto the system. In fact, all of the pages (i.e., the entire data base) is sent to all users every

15 or 20 seconds usually at high speeds such as 9600 or 19200 baud. If a user asks for a page (e.g., the foreign exchange page for a bank), it takes at most 20 seconds to display that page on the screen. But what is more interesting is that once a page is on the screen, it gets an automatic update every 15 or 20 seconds, since the screen device continuously polls the incoming stream of pages and can pick off the page that the user is currently requesting. There is virtually no local memory in the screen device, nor is there much local computer intelligence.

This type of service does not have the overhead of two-way city-to-city networks, since all information is sent on a one-way basis only. There is no need for error correction. Using checksums, a record can be observed to be at fault at the receiving station, and ignored. Within a few seconds, a new record containing the same (or updated) page is available, so the loss of a record due to a transmission error is not serious. Billing is made simple by charging a flat monthly fee for the service, and due to the simplicity of the service the monthly fee ends up being quite reasonable, considering that you can use the service 12 hours a day if you want. A schematic of such a one-way network follows:

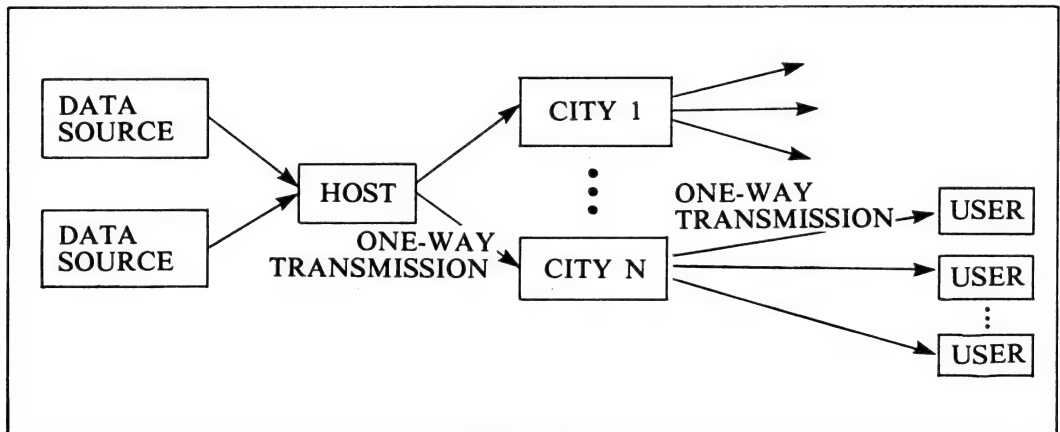


Figure 2

The advantage of this approach is that the addition of a new user represents no additional drain on the resources of the central host computer, and that the speed of retrieval and screen refresh is high. This type of technology has become one of the most popular methods of information dissemination used in the world today.

One disadvantage of one-way networks like this is that every time you want to add more information, you have to increase the cycle time for one transmission of the entire data base. It will be a few years (but perhaps not many) before such a system could transmit all 50

billion bytes of the I. P. Sharp public data bases in 15-20 seconds! The other significant problem is that there is no way for a user to communicate with the host computer. This rules out many useful applications, the most obvious one being inter-user electronic communication.

Since 1980, a number of new forces have emerged which have affected—and will continue to affect—the technology described above. These are satellite transmission, the microcomputer, and improvements in storage technology. Any one of these new technologies is capable of having a significant impact on the way information is distributed and presented to the end user. The fact that all of these technologies are emerging at the same time provides a powerful explosive mixture, the result of which is not easy to forecast.

Satellite technology permits a data source to broadcast large amounts of data to anyone who is able to receive it, at a reasonable price. Data rates may vary from 300 to 19200 baud. For example, the cost for a data provider to transmit a continuous 1200 baud service from the Westar IV satellite is \$10,000 per month. The purchase price of a receiving station (a 24 inch dish and associated hardware) is under \$3000. A network connect fee of \$10/month is usually charged. For one-way networks such as the one described above, this is an ideal way to reach customers, especially those who are not resident near a large city (e.g., mid-Western U.S. commodity traders.) Since there are no communications wires to connect, it's easy to get a new user started. A drawback is that in crowded downtown city centres, a user does not always have an unobstructed path from the receiving dish to the stationary satellite. And of course, these are still one-way systems, so users cannot send signals from their station to the host, or to another user. Such a distribution method can be depicted as follows:

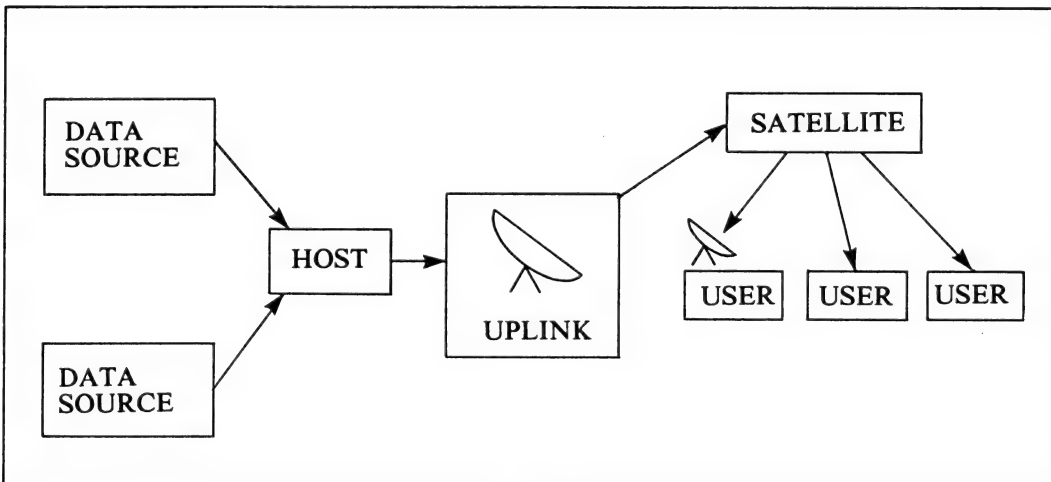


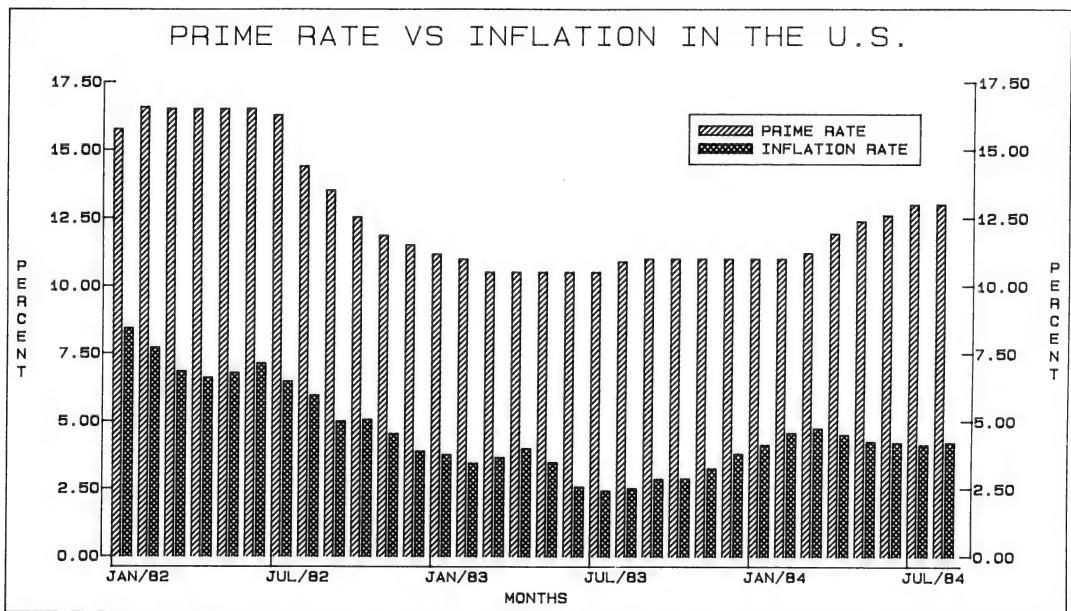
Figure 3

Looking at the impact of the microcomputer, the possibilities begin to multiply. Let us look at what the microcomputer means to a typical data base user on the two-way I. P. Sharp network. Historically, a user who wanted to plot interest rates and inflation would have retrieved and plotted the data using only mainframe CPU cycles. But only about 25% of those cycles involve retrieving the data. The rest of the cycles are used to manipulate the data, and to perform the plot. The MAGIC statements required to accomplish the plot are:

```

RESETOPTIONS
MONTHLY, DATED 1 82 TO 8 84
SETPLOTOPTIONS A DETAILS NOT SHOWN
PUT CITIBASE 'FYPR,PUNEW'
PUT 12 PCHANGE ITEM 2
PLOT ITEM 1 3

```



CITIBASE/I.P. SHARP ASSOCIATES

Figure 4

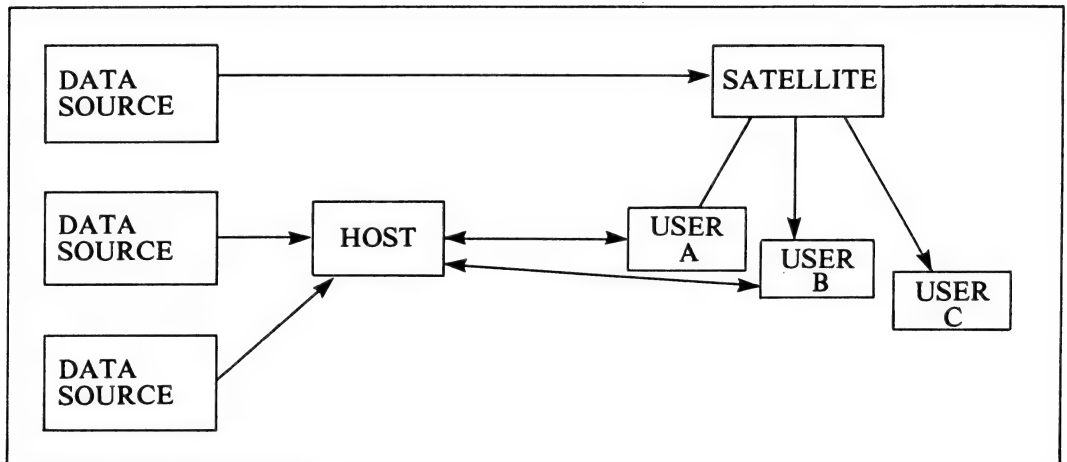
With MAGIC available on a PC/370, all of the above statements can be executed on the PC, except for the data retrieval (i.e., *CITIBASE 'FYPR,PUNEW'*). Most users of data bases believe that this transfer of cycles away from the mainframe over to their local CPU should substantially reduce their timesharing bill. But from a data provider's point of view, it should be pointed out that the cost of mounting and maintaining these data bases is not significantly reduced by the availability of microcomputers. Thus, the use of micros to retrieve data from a public data base may force a change in the method of billing for such access (such as a per item access fee).

In the above scenario, the micro is used to provide CPU cycles, but the data is still presumed to reside on the mainframe. As storage technology improves, it seems reasonable to keep local copies of data bases (such as a stock price data base). But it will probably be several more years before this concept works, for both technical and practical reasons. First of all, these data bases are quite large (e.g., 400 million bytes). Updates must be performed at regular intervals (e.g., daily for a daily stock price data base). A certain amount of manual maintenance is always required (e.g., adjusting for stock splits, performing error checks, etc.). For these reasons users might prefer to retrieve their data from a centralized source. But in the future, it might be possible to perform the updates to a microcomputer which has several billion bytes of local storage and a fast processor. It would have to be quite fast, for typical daily updates for U.S. stocks on the I. P. Sharp IBM 3081 mainframe can consume over an hour of elapsed time. What is needed is a processor on a micro that is about 10 times faster than the current IBM 3081. That is not likely to happen in the next 5 years.

Micros, of course, have a significant impact on the performance of one-way broadcast networks. By placing a micro on the end of a one-way transmission, the micro can store, and keep updated, everything that is sent to it. When a request for information is made, the micro is capable of delivering the requested material from local storage immediately. Private corporations may one day find that this is the most effective way to deliver relevant corporate information to their managers. Assuming infinite storage availability at the local micro, the data processing centre of a corporation could frequently (e.g., once per day) transmit its entire corporate data base by satellite to receiving micros. (Security is one issue which can be adequately provided for by satellite transmission, but is not covered here.) A manager may require a particular piece of the corporate data base at any point in time. It is instantly available. If it is not required, who cares? The point of this is that whether there are 10 or 1000 users of the corporate data base, there is no difference in the hardware required of the host computer to deliver the information users want. And data retrieval is fast.

As mentioned before, this approach is lacking in that it is not possible for a user to send data from the local workstation to the host. In many instances, this is not a problem. But

there are many applications that do require input from users. It would appear that the ideal workstation involves a combination of the two technologies. Such a system is illustrated below:



**Figure 5**

In this case some of the users (like user C) do not need to be connected to the host computer. The one-way receipt of information via satellite is enough to satisfy all of their computing needs. If user C does need to communicate with the host (as do users A and B) an automatic connection to the host for this purpose is accomplished. This sort of combination system appears to have all the advantages of both capability and cost, and is quite possibly representative of the technology that we can expect to see in the next few years.

The title of this session is "Managing the Information Revolution". This paper has covered the technical revolution, but has spent little time discussing about how to manage it. The following case studies provide more insight into how other corporations are currently managing their information needs, and what their plans are for the future. Those companies that are best able to understand the new technology and harness it successfully are quite likely the same companies that will survive and prosper in the information world of tomorrow.

## Distributed Intelligence in Application Network Design

James L. Schmit, Director of Research and Development, Computer Assisted Analysis

### BACKGROUND

The last decade has seen numerous developments in computer hardware and software that directly affect application systems design. The trend has been toward smaller, more powerful, and less expensive components. Some have chosen to turn these advances into improved versions of existing designs, either cheaper units of equivalent functionality or more powerful units at equivalent prices. Others have created new designs that were not physically or economically possible before such components. A good example of the former tack is the super minicomputer; of the latter, the personal computer.

The components that have had the greatest impact on hardware system design are the microprocessor and the high density dynamic ram. Together they permit the construction of powerful single user computer systems at prices in the range of better terminals. The existence of such hardware has encouraged the development of software systems that have turned a generation of professionals into computer users.

The most common manifestation of the new technology is the personal computer (PC). In its usual form the PC exists as a complete standalone single-user computer system. As such, it has represented a significant economic challenge to centralized computing. Totally decentralized computing has two significant shortcomings, however: reduced communication, and a major data processing chore with respect to its local file system.

An alternate design that attempts to combine the best features of centralized and independent computing is the intelligent workstation. It utilizes local application program execution with centralized data bases. The workstation may have its own file system, but stores only locally relevant information. This approach greatly reduces the processing and communications burden on the central system. This paper will demonstrate the efficacy of the workstation approach through an application example. The application is a technical analytic market analysis system known as the Compu Trac d (pronounced 'delta') Trade Plan system.

### APPLICATION

Analysis of markets, with an eye toward trading profits, can be divided into two schools. Fundamental analysis is the study of current factors, either physical or financial, that may have a future market impact. It is often qualitative in nature. Technical analysis examines



past market action to indicate future trends. It is extremely quantitative and most often depicted graphically. The steps in the technical analytic process are:

1. Data collection of market prices and activity.
2. Data reduction through calculation of technical indices.
3. Data presentation through charting.
4. Modeling by the selection of a relevant set of indices.
5. Simulation to determine optimum trading strategies.

Computer Assisted Analysis (CAA) has been in the business of supplying computer tools to speed the work of the technical analyst since 1978. It has historically produced two types of systems which are marketed under the names of Compu Trac and Intra Day Analyst. Compu Trac systems connect with market price data bases at the end of each trading day to update a user's local data base. All analyses then use the local data base. Intra Day Analyst systems perform a similar analytic function but operate in real time through a connection to a market price reporting network. Versions of both systems have been produced for the Apple II computer and IBM PC. CAA has delivered 3500 such systems worldwide. The Trade Plan system is a natural evolution of the products using CAA's own internally developed hardware and system software to produce an intelligent user's workstation. The main design objectives of Trade Plan were ease of use and flexibility.

## **USER'S MODEL**

A user interacts with Trade Plan through a PC type keyboard and a high resolution colour display. The display is used to implement a desktop metaphor. The user commands the system by manipulating objects which appear as sheets of paper on the desktop. The display is related to the keyboard by representing the 10 function keys as a keypad on the desktop. The definition and use of the keypad is dynamic and the system is driven whenever possible with a single keystroke. The bottom two keys are reserved for special functions. The HELP key is always active for user assistance. When struck, a sheet of paper appears on the desktop describing all options. The next keystroke removes the help sheet. Beginners may request a GUIDE which automatically provides help at appropriate points in the system. The TOOL key allows the user to temporarily suspend his activity to use one of a set of system functions such as calculators and references. Figure 1 is an example of a screen display with a HELP message.

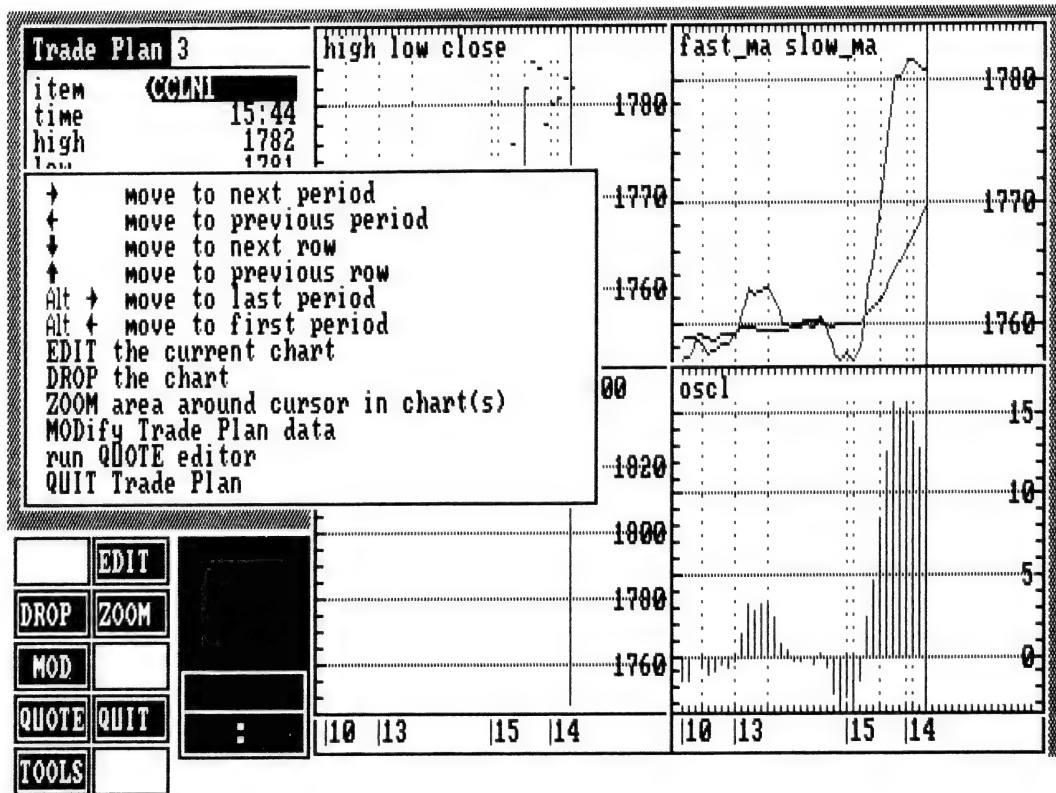


Figure 1

Technical analysts traditionally work with two types of documents. A spreadsheet is used to record price data and to systematically calculate indices. Once calculated, the results are transferred to a set of charts for inspection and interpretation. Trade Plan has made these documents the central objects in its system.

The Trade Plan spreadsheet is a collection of market data, coefficients, predefined technical indices, user defined indices and trading simulations. It functions in a manner similar to electronic spreadsheets but takes advantage of the special vector nature of the data to ease definition and use. The size of the spreadsheet can vary, with a typical dimension of 32 rows by 128 columns permitting a very involved analysis of 6 months of daily data. The entire sheet can be examined within its display window by cursor movement. Elements of the sheet can be altered forcing the recalculation and display of affected items. Changing the name of an item under analysis, for example, will force a reload of new data with a complete recalculation of all indices. A single Trade Plan can be used to analyse a wide range of stocks or commodity contracts. Figure 2 shows a Trade Plan spreadsheet.

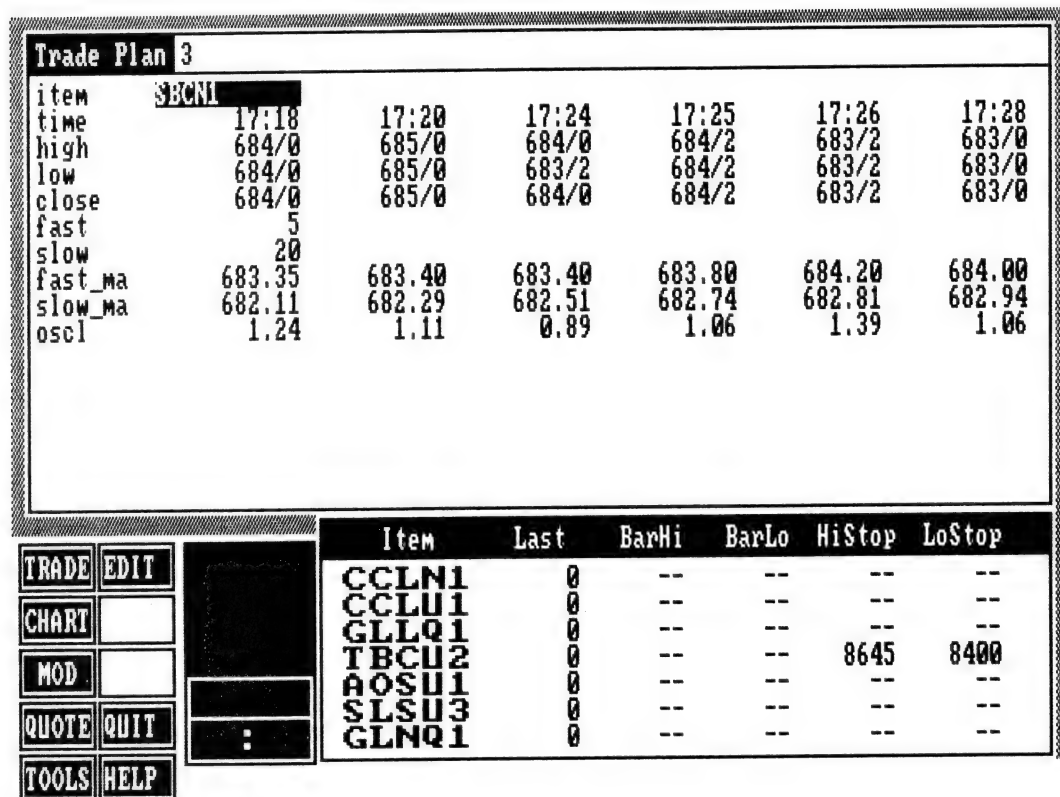


Figure 2

Trade Plans are user-defined with a considerable amount of system assistance from a special editor. Figure 3 shows the definition of the Figure 2 plan.

Name	Definition	Decimals
item	"SBCN1	
time	DATA(time,item)	
high	DATA(high,item)	
low	DATA(low,item)	
close	DATA(close,item)	
fast	5	0
slow	20	0
fast_ma	Mov_avg(close,fast)	2
slow_ma	Mov_avg(close,slow)	2
oscl	OscI(fast_ma,slow_ma)	2

ADD	DEL		Item	Last	BarHi	BarLo	HiStop	LoStop
REPL	CLEAR		CCLN1	0	--	--	--	--
			CCLU1	0	--	--	--	--
			GLLQ1	0	--	--	--	--
			TBCU2	0	--	--	8645	8400
	EXIT		AOSU1	0	--	--	--	--
		:	SLSU3	0	--	--	--	--
TOOLS	HELP		GLNQ1	0	--	--	--	--

Figure 3

Charting is performed by the system to the user's specification. A charting sheet can be divided into from 1 to 10 chart regions. Each region can display up to four plots of a wide range of types. Three time axis spacings are provided. Figure 4 shows a typical charting of a spreadsheet. The chart overlaps the spreadsheet but leaves a single column exposed for examination. A graphic cursor in the chart is synchronized with the spreadsheet cursor to allow the user to examine the data that underlies the chart. Changing a spreadsheet element will again force the recalculation of all affected items along with the redrawing of all altered charts.

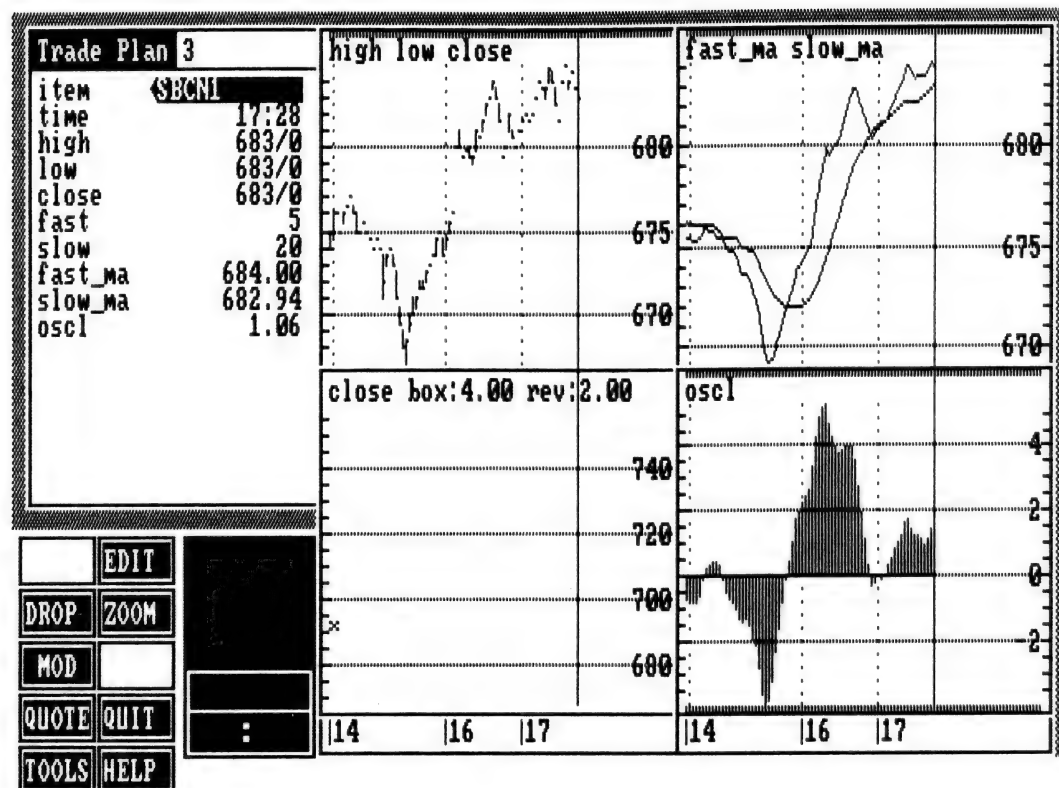


Figure 4

Charting is specified with a special editor in a manner similar to the spreadsheet. Figure 5 shows the user definition of the chart in Figure 4. Special tools are provided to permit the marking of charts with labels and trend lines. All spreadsheet data and charts can be printed in colour.

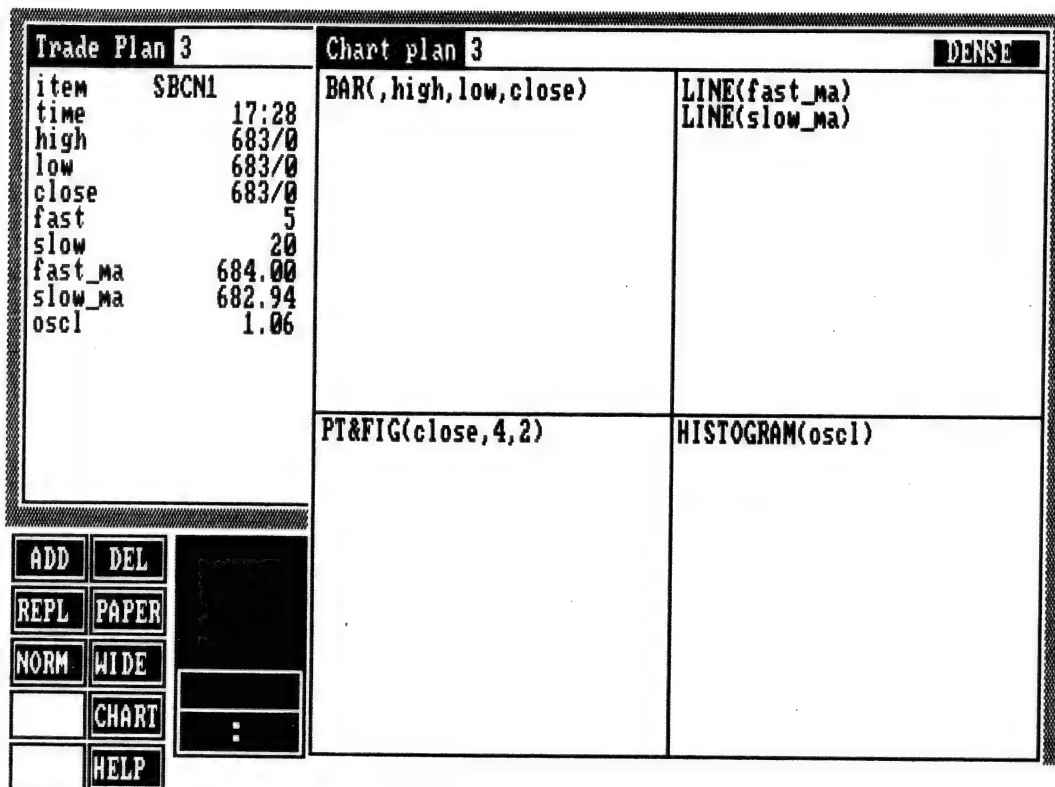


Figure 5

A unique feature of Trade Plan is the integration of trading strategies into the spreadsheet. The system then traces the economic consequences of the trading system. Dynamic alteration of the control coefficients coupled with graphic depiction of the results permits interactive simulation and system optimization. (See Figure 6.)

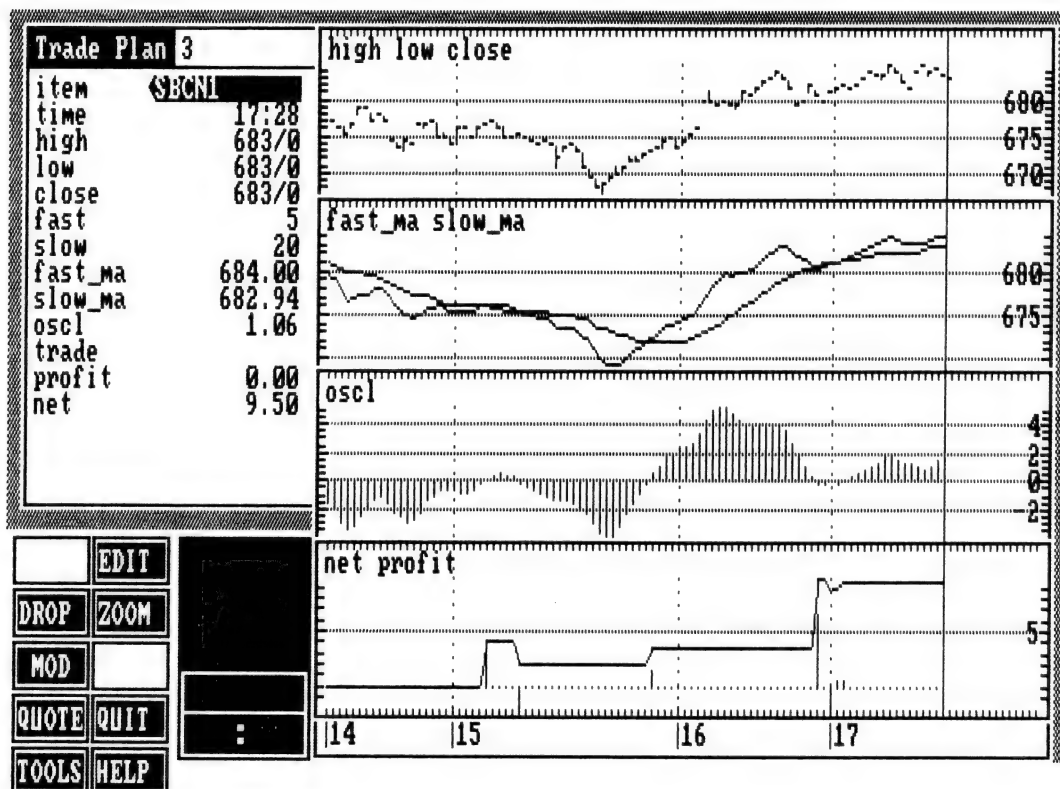


Figure 6

## INTERNAL DETAILS

Trade Plan is implemented as two major components, a central system function performed by the I. P. Sharp computer system in Toronto and a local function performed at the user site. The two functions communicate over IPSANET, the I. P. Sharp Communications network. A major design goal was to minimize the burden placed on IPSANET by Trade Plan, thus reducing user cost.

The central computers in Toronto store the I. P. Sharp price data bases that contain both historic market activity data and real-time prices. An APL function was written to make that data available to user systems on command. The function can both respond to a specific request and generate unsolicited data transfers at user specified time intervals. Responses are transmitted as packed binary records with checksums to ensure transfer integrity.

The local user activity is implemented on a CAA designed intelligent user workstation with CAA-designed software. Local hardware consists of a delta computer with 256 to 1024 K memory, 2 serial i/o channels, a 640 × 240 8-colour bit mapped video display and a local

disk-based file system. An optional ink jet colour printer for hard copy of all text and graphics is usually also included.

The systems software used by the delta computer is based upon a function-oriented reverse Polish notation architecture known as the delta pseudo machine. Functions are written in an RPN structured language called d (delta). The function-orientation makes no distinction between language and operating system. The delta program development environment was designed to achieve three major goals: efficient object code execution, extreme object code density, and minimum program development time. The efficacy of the design is demonstrated by the Trade Plan application which contains less than 48 K of object code and was developed by a single programmer in less than 3 months.

Trade Plan is implemented on the local workstation as 4 concurrent tasks with interrupt driven i/o routines. The i/o interrupt service routines all either fill or empty ring buffers and are not considered tasks. The highest priority task is the communication line monitor. Initiated by a 100 msec interval timer interrupt, it extracts characters from the communication ring, unpacks the fields and verifies the checksums. The line monitor posts certain state variables but primarily places line commands in a command ring for processing by the next task. The command processor, the second highest priority task, extracts and processes records from the command ring. The command processor posts period high and low prices, controls system functionality through download permissioning, and determines the timing of history vector update and backup. If an update or backup is in order, it captures relevant data and places a request in an update ring. The lowest priority task is the update processor which takes its commands from the update ring and performs the desired function. The user task is the second lowest priority task in which the spreadsheet and charting functions operate. User keyboard input is taken from a typeahead ring but screen output results in immediate modification of the video bit map.

To achieve the goal of true ease of use, Trade Plan places no data processing or file backup responsibilities on the user. The local disk file system was viewed as nonvolatile storage for 3 classes of information: application function code, Trade Plan specifications, and market history data. Function code resides on the disk but being static in nature poses no update/backup responsibility. The extreme code density of the delta machine coupled with the communication capabilities of IPSANET does permit code replacement and update should it prove necessary. Trade Plan specifications are handled in a simple but effective manner. The system can store 8 different plans which can be user-modified. The act of modification, however, results in the automatic storage of the most recent specification. The user always finds each plan in exactly the state in which he left it without a specification library maintenance chore.

The handling of market history data is a radical departure from past designs which maintained a local data base with its maintenance responsibilities. History data in Trade Plan is kept as a cache to reduce network demands. Traders tend to analyse the same items on a day to day basis. The system caches the last 128 histories requested. A new request need



only fill in the data activity since the last request. All caching is transparent to the user. A partial, or even complete, loss of cached data will result in no user inconvenience beyond a somewhat slower initial response time. Backups are not required. All communication and file activities are user invisible.

## **CONCLUSION**

The rapidly decreasing cost of computer power has tipped the scales in favour of local processing in a large number of applications. The value of a computer network with a central system should not, however, be overlooked. The intelligent user workstation can be a highly cost efficient method of expanding the functionality of traditional central systems. Its use should be considered by all application system designers.

# The Information Revolution at Morgan Stanley & Company

Keith W. Iverson, Morgan Stanley & Company, Inc.

## INTRODUCTION

Morgan Stanley is an Investment Bank serving Fortune 500 companies and large institutional investors. We provide our clients with unique Investment Banking, Corporate Finance, and Capital Market services which depend on our ability to analyse and condense vast quantities of data into sound and creative client recommendations in the area of recapitalization, pricing/timing of new equity/debt issues, and trading strategies in the capital markets arena.

MIS at Morgan Stanley is led by a Managing Director who is a full partner in the privately-held firm. Three key MIS positions are filled by Principals who also participate in the equity of the firm. These three key positions are divided among the Systems Development, Analytical Systems, and Technical Services groups. The first two are application development groups and are each divided into special projects and client coverage teams. From the MIS perspective, clients are other departments in the bank. Client coverage teams are responsible for providing all computer services to their respective client areas. Technical Services comprises Data Center Management, Systems Programming, Communications, and Data Administration.

The Technical Services computing strategy is two-fold: (1) to provide the end user with a single entry point to all computing services; and (2) to provide the application programmers with maximum flexibility in choosing the development environment most appropriate for any given application. The computing environment at Morgan Stanley is complex, with multiple CPU's, an extensive communications network, two major development languages, and two major data base management systems. Microcomputers are beginning to have an impact on the user perception of computing services and must be integrated into the computing strategy.

Data Administration, part of Technical Services, is responsible for the management of data facilities and services within the strategy as defined above. In this context, Data Administration works closely with the other Technical Services Groups in implementing the strategy. This paper focuses on the computing environment and the information requirements at Morgan Stanley from a Data Administration perspective while discussing some of the problems we face in implementing the two-fold computing strategy and the efforts underway to resolve those problems.

## **THE MAINFRAME COMPUTING ENVIRONMENT**

On the desks of 60% of the employees in the firm are IBM 3270-type terminals which are connected into a single data centre located in the Wall Street area of New York City. The network includes the branch offices in San Francisco, Chicago, Toronto, London, and Tokyo. The data centre consists of 4 large mainframes (1 IBM3084, 2 NAS9080's, and 1 NAS9060) with an IBM 4341 network frontend, and 100 billion bytes of online 3380 DASD fully interconnected to all CPU's. The VTAM network, with some specialized proprietary Morgan Stanley software, allows each end user to run multiple sessions connected to any combination of mainframes required. For example, a user from a single 3279 graphics terminal can have active a CICS trade-entry application, a CICS accounts-intention application, and an APL application. The user switches from application to application with only a few key strokes.

Traditionally, MIS has segmented the client areas into back-office and front-office operations. The back-office encompasses the brokerage activities associated with accounting, such as maintaining trading and commission accounts, producing client statements, and keeping the books on new issue offerings as well as general ledger and payroll systems. Back-office systems have generally been developed and supported by the Systems Development Group. Front-office operations are the sales, trading, Mergers Acquisition, and Corporate Finance activities. Applications development and support for the front-office is generally provided by the Analytical Systems Group (ASG).

Applications are written in one of two languages, Software AG's NATURAL and SHARP APL. NATURAL is used almost exclusively to build back-office transaction processing systems which feed the data bases maintained in ADABAS (a data base management system, also from Software AG). ADABAS encompasses about 45 billion bytes of data. APL is used almost exclusively by the Analytical Systems Group in building applications for decision support in the front-office. The data bases required by the ASG are in HYDRA, a Morgan Stanley-built data base management system in SHARP APL. APL controls about 15 billion bytes of data. Morgan Stanley is one of the largest installations for SHARP APL and for Software AG's NATURAL and ADABAS.

Further complicating the mainframe environment is the use of microcomputers by end users. This newer technology requires integration into the network architecture and access to the mainframe data bases. The issues surrounding microcomputers will be dealt with later in this paper.

## **THE CHANGING INFORMATION REQUIREMENTS**

In the past, the back-office and front-office requirements were mutually exclusive; therefore, using different data bases and development environments was not a major problem. The needs of the business units are becoming more sophisticated, requiring access to combina-

tions of data stored in ADABAS and APL. The two examples below demonstrate the need for more robust data access facilities to bridge the application and data base environments. The back-office has an application called fluid segregation which allows the bank to use a portion of the stocks held in customer margin accounts. The task is to maximize return from the use of these stocks without violating the regulations of the Securities and Exchange Commission. Data for this application is generated from other back-office systems and stored in ADABAS. APL, however, was needed to perform the analysis within the seconstrains. This is a nightly job so a NATURAL program writes an OS dataset which is read by APL to perform the final data preparation, analysis and production of the fluid segregation reports.

Front-office operations are becoming more sophisticated and demand that back-office information be accessed directly by analytical programs preparing personalized services and presentations for their clients. This requires APL-based analytics to access back-office data from ADABAS. For example, the sales people in the Corporate Bond trading area are responsible for notifying their clients of opportunities in the bond markets. To do this, they need applications to access data from both ADABAS and APL. A sample table of the required data and sources are listed:

<b>Data</b>	<b>Source</b>
Client portfolio held at Morgan Stanley	ADABAS
Client holdings with other brokers	ADABAS
Current bond bid/offer quotes	APL
Market and cross-market trends	APL

Starting with general market trends, trading strategies, and known blocks of buy/sell orders, the salesperson identifies clients for whom buying or selling specific bond holdings at specific prices will have a positive impact on their total known portfolio. The analytical tools needed by the sales people require online access to data spread across both ADABAS and APL. Data Administration built a generalized APL interface to ADABAS allowing implementation of the application in APL, providing online access to all data available in both data base environments. The ADABAS interface in APL is discussed in more detail in the following section.

## **DATA ADMINISTRATION DEVELOPMENT EFFORTS**

The two examples cited above are the just the beginning. Data Administration is actively seeking creative interfaces between the development and data base environments. Our objective is to provide maximum flexibility in choosing the correct language to implement new applications being demanded by the business units of the bank. Data Administration efforts are being concentrated in four areas: (1) facilities for batch transfers of data between ADABAS and APL; (2) a general interface for APL access to ADABAS; (3) a general

interface for NATURAL programs to access APL data bases; and (4) a re-evaluation of the location of specific data bases in one environment or the other.

1. Toolkits for batch data transfer have been provided by Software AG, I. P. Sharp Associates, and development work at Morgan Stanley. We now have facilities which load data from and into APL and ADABAS files using flat OS datasets as the common transfer medium. The application code on both sides must know the file layouts in the OS dataset and we need to provide facilities which transfer field and data-type information along with the flat files.
2. Data Administration recently implemented an interface in APL which provides the APL user or applications programmer access to ADABAS. ADABAS is designed as a transaction oriented data base manager. It was a challenge to provide an APL interface oriented towards arrays and APL-like language access facilities similar to those available for the APL data base system. The key was to map arbitrarily complex APL selection criteria into the necessary sequence of ADABAS commands which are then submitted through the SHARP APL FCAP auxiliary processor to ADABAS. It is interesting to note here that, in our shop, ADABAS runs on multiple CPUs and data bases are segmented across these CPUs using channel-to-channel connectors for access to any of the data bases from any of the CPUs. Using the interface we implemented, APL accesses ADABAS on a single machine and ADABAS provides access to any of its data bases on any of the other CPUs.
3. An interface from NATURAL to APL data bases poses a series of complex problems which we have not resolved. The major problem is the ability to access APL data bases on one CPU from other CPU's. Once this problem is resolved, it is theoretically possible to implement a toolkit which runs as an NTASK in SHARP APL and gets data requests from NATURAL programs through an auxiliary processor which would have to be custom built. The primary complexity in the APL NTASK application is multi-threading requests from a number of simultaneously running NATURAL programs across a number of APL data base files including such facilities as file and record lock-outs with updates and transaction commitments.
4. The problems faced in providing an interface from NATURAL to APL data bases have led to investigating the possibility of relocating selected APL data bases into ADABAS. APL applications requiring access to these data bases would then use the APL/ADABAS interface discussed earlier. The advantages to having a single data base management system in the shop are compelling and since the APL interface to ADABAS allows the user to view the data in an array oriented manner, the barriers to implementing such a strategy would appear minimal. Our investigations show that there are some basic conceptual differences between the data we manage in ADABAS and the data we manage

in APL that complicate implementation of this strategy. Data bases in APL support analytical applications which are heavily biased towards the use of time series data. ADABAS provides no time series facilities. In fact, the record length restrictions in ADABAS will not allow the storage of time series fields such as daily pricing information for a stock over a six year period. Tackling the problem from a different perspective suggests that each record in the file becomes a time point. This approach would cause some of the data bases to exceed the ADABAS 16 million record limitation. Another major conceptual difference is that in the APL data base system, any field in a data base file can be used as a key for selection and in fact, many APL screening applications depend on this facility. ADABAS on the other hand allows selection only on fields that are defined as keys and, for efficiency reasons, the number of keys for a data base should be limited. Data Administration is experimenting with different ADABAS file designs and putting up test versions of some of the APL data bases in ADABAS in an effort to resolve these problems.

## **MICROCOMPUTERS**

Microcomputers are now being used in the offices of banking professionals. The proliferation of microcomputers throughout the firm will have a dramatic impact on the way data resources are managed, requiring the control of distributed data bases to ensure that all data required by each area is available in an accurate and timely fashion, while eliminating unnecessary duplication.

Business units initially purchased microcomputers without support from the MIS groups. Early in 1983, MIS recognized the utility of microcomputers and became actively involved in the support and integration of microcomputers into the overall computing environment. There are now about 150 IBM PCs and 50 APPLES in the hands of users. Current installation rates are averaging 10 per month. Microcomputers are primarily used at Morgan Stanley for word processing, spreadsheet, and departmental data maintenance. The microcomputer users are realizing that much of the data they need to manipulate is on the mainframe computers and that their application needs sometimes grow beyond the capabilities of the microcomputer.

Our efforts to date with microcomputers have been directed at becoming familiar with the technology, selecting hardware and software, managing the placement of micros, keeping track of the inventory, and trying to integrate microcomputers into the computing strategy at Morgan Stanley. We are working directly with the designers and manufacturers of microcomputer products and building in-house programming expertise, and are now planning the development of some of the software required to meet the integration needs of microcomputer users.

We do not necessarily view the microcomputer as a personal computer, but more as a

computing resource to be integrated and offered as part of our personalized computing environment. The major problem we see with microcomputers, as with any other new computing facility, is the delivery to the end user. If with each new computing facility we were to place another terminal device on the user's desk and pull another set of wires, an already complicated environment would become unwieldy. How many screens, printers, and keyboards can one user deal with? Our strategy, as stated earlier, is to have one device on a user's desk with access to all computing facilities. We want this to include microcomputers. We are experimenting with several options including IBM 3270 PCs and shared PCs hooked into our SNA VTAM network.

The IBM 3270 PC solution is attractive on the surface, but our experience has been less than successful. The IBM 3270 PC does not support the APL character set. The screen characteristics, the clarity, and the stability of the display, leave a lot to be desired. Mainframe GDDM graphics are not supported. The keyboard is cumbersome. The device has a relatively large footprint and cannot be used as a shared resource. The recent announcement from IBM on their IBM 3270 PC/GX will correct some of these perceived deficiencies; however, the device will remain as a non-sharable resource.

Morgan Stanley has invested a lot of resources into the replacement of the large IBM 3270 devices with smaller DAVOX equivalents. Direct work with DAVOX has resulted in their production of an IBM 3279 equivalent terminal with full colour, APL, and GRAPHICS support. DAVOX terminals require the installation of DAVOX 3274 equivalent controllers which have the added capability of hooking on IBM PCs and offering them as shared resources to all DAVOX terminals hooked onto the same controller, each PC being able to support only one user at a time. Additional facilities being developed are online and batch data transfers between the DAVOX-connected microcomputers and the mainframe computers. The experimental configuration we are working with includes a PC/XT as a file-server and several IBM PCs attached to the same controller. The advantage to this approach over a local area network is that no additional cabling is required to provide access to users. A single DAVOX terminal on a user's desk provides access to all mainframe facilities, including graphics, as well as access to microcomputer applications.

We do not expect that the DAVOX connected PCs will satisfy all user requirements for microcomputers. There will still be quite a number of standalone units throughout the firm. To support the needs of these users, we are designing and implementing mainframe data access facilities. The facilities planned include: (1) the ability to use the mainframe as a store-and-forward switch for users who need to transfer data between microcomputers; (2) the ability to screen and selectively move data from mainframe data bases to the microcomputer for use in spreadsheet and data base applications; and (3) online query access to mainframe data bases from microcomputer applications which we anticipate building in the future.

## **SEBPOST: Computer Mediated Human Communication**

Gösta Olavi, Senior Systems Analyst, Department for Marketing and EDP Development, Skandinaviska Enskilda Banken

### **BACKGROUND**

Skandinaviska Enskilda Banken (S-E-Banken for short) is one of the two largest commercial banks in Scandinavia. The S-E-B Group also contains several international subsidiaries and agents' offices. The Department for Marketing and EDP Development plays an important role in conceiving, and implementing, new products for the highly competitive national and international banking markets.

In the Fall of 1981 the bank began to show interest in electronic mail, prompted by several products which became commercially available at that time. Also, computer conferencing was used in a pilot project within EDP management, giving some first contact with this new means of sharing ideas.

It was, as usual, a long way from interest to a useable product. Many products in this area are developed in environments other than IBM/MVS/VTAM. These are very difficult for us to install. Most are also difficult to tailor to the specific needs of an installation, e.g. translating the user interface into Swedish or reassigning PF-keys.

As it happened, S-E-Banken was in the process of evaluating a new and better APL system, having outgrown the old one. While examining SHARP APL, the MAILBOX product was seen as a possible electronic mail medium. It became clear that it had most of the basic features that we needed. It was also estimated that, with reasonable development resources, a superstructure could be built to give, for example, the required user interface, and also conferencing facilities.

The availability of MAILBOX was, in fact, one factor in the decision to move to an in-house SHARP APL system in 1983.

### **ELECTRONIC MAIL AND COMPUTER CONFERENCING**

The benefits of electronic mail over more conventional communications media such as the telephone and paper mail are well known, especially among users of the SHARP APL system. Several products are available in this area, often within international networks.



Some examples are the PTT's Telex networks, systems by Tymshare, General Electric, and the Computer Corporation of America.

So what is computer conferencing about?

Among the characteristics of mail systems is that they put much control in the hands of the sender of a text. The sender decides who is to read the message and expects mail to be read; and to be read within, say, 24 hours. Often this is for good reasons. In many cases, however, texts are comments or opinions to other texts, and not vital for everyone on the distribution list. With a growing number of users (and an increasing use of distribution lists) many receivers will meet an information overload problem. They get too much in their in-tray, and have little help in making intelligent selections among the incoming texts.

In a conferencing system, many texts are not primarily sent to individuals but to 'conferences'—or meetings, discussion groups, interest groups, or whatever they are called. Thus, a conferencing system moves some of the control from the sender to the receiver. The receiver can decide which discussion groups to read about today, and even which discussion groups to join.

Also, a conferencing system has automatic archiving of what is sent to the discussion groups, plus flexible tools for retrieving texts. The archiving function is generally regarded as the distinguishing feature which separates conference systems from simple mail systems. The archive contains a complete transcript of what has been contributed on a specific theme. Users who join a group late can easily get up-to-date with the discussion, without disturbing those who have already read the earlier texts. Most conference systems also have a form of 'sub-conferences', recapitulating a sequence of comments that followed from a certain text.

Conferencing systems have often emerged in environments where interest is shown in the user dialogue. Often these systems offer a range of dialogue schemes, from a strictly menu-driven approach to aid the novice user, to a flexible command language with user-defined procedures.

Conferencing systems often require features which are found in mail systems. Good conferencing systems include features for person-to-person texts, private archive files, commenting and answering of texts, and other features that good mail systems offer.

As noted, electronic mail is used instead of, or complementing, paper mail and the telephone. Conferencing has other aims as well—it is used as a new way of sharing ideas, sometimes even with people you have never previously met. To some extent it can replace group meetings—or be used to prepare and conclude them. Also, it is often used for joint authoring of larger texts—some systems contain 'notebooks' to help with this.

More on electronic mail and computer conferencing can be found in references (1) and (2).

## **SEBPOST FEATURES**

SEBPOST is based on the MAILBOX product and supports most of its options. These are however imbedded in a superstructure that also supplies conferencing facilities, a simple command language, and online help.

The following will give a quick glimpse of SEBPOST, and of how it differs from the MAILBOX base product. A more detailed description can be found in reference (3).

### **Reading News**

When the user enters the system, a summary of incoming mail is displayed, grouped by destination:

- private
  - to a certain distribution list
  - to a certain discussion group

The user can select texts according to

- destination
  - sender
  - urgency
  - internal text number, if that is known

The selected texts can be viewed

- one at a time
  - continuously by discussion group
  - continuously all texts

There is also a SUMMARIZE facility that displays the SUBJECT line from the selected texts.

While displaying a set of texts, the user can for each of the texts:

- answer it to the author
  - comment it to all original recipients
  - Forward it to a colleague
  - assign search keywords and archive it in his private archive file
  - hardcopy it
  - Recapitulate the comment chain leading up to it
  - or simply delete it

### **Discussion Groups**

All users are allowed to start discussion groups (conferences). These may be 'open', in the sense that everybody can enroll themselves; or 'closed', where only the group coordinator can enroll participants. Discussion groups normally have central archiving.

### **Retrieving from Archive Files**

Texts can be saved in personal action files, or in discussion group archives. To retrieve texts, a user can then name his private archive, or a discussion group. In addition, any combination of the following fields may be used:

- sender
  - receiving user (or group)
  - sent before or after a certain date
  - search keywords (personal file only)

### **User Interface**

The user interface in SEBPOST is a simple command language. Most commands are available in the main dialogue loop. Any parameters missing are prompted for by the system. The HELP facility is available at every point in the dialogue. Often a second help request gives more extensive information than the first.

The experienced user can utilize a typeahead input buffer, answering anticipated prompts in advance. User-defined procedures (like saving these typed-ahead inputs) are, however, not supported at present.

Full screen 3270 processing is used in some situations:

- editing a text

- starting a new discussion group
- retrieving texts from an archive, with various search criteria

Apart from this, a line-by-line dialogue is used, employing features of the IDSH session manager.

Commands to SEBPOST, and messages from the system, may be in a number of different languages, selectable by the user on the fly. Swedish, English, Dutch, and German are available now, and any careful translator can extend this to the language of his choice.

## **IMPLEMENTATION HISTORY**

SHARP APL was installed at S-E-Banken in June, 1983. Almost immediately, MAILBOX began being used in parts of the EDP department. Most users preferred a simple superstructure, including a full screen text editor, to using the native MAILBOX workspace.

During the fall of 1983 a prototype of SEBPOST was developed, and enjoyed limited use within and outside the EDP department. It provided a simple user menu, and supplied full screen text editing. Archiving for the conference-like features was provided by having a dummy user enrolled as a member of most distribution lists, archiving all its incoming texts into central files.

With the advent of MAILBOX version VIII, work could start on the production system. The base product now introduced Special Interest Groups; group FILE messages; the SUBJECT line that can be added to texts; and processing functions which return explicit results, allowing the superstructure to be reasonably complete. All these features are needed in a conferencing system. MAILBOX, together with the Message Archiving and Retrieval System (MARS), provides the foundation for SEBPOST.

SEBPOST release 1 was constructed during late 1983 and the spring of 1984, with most work being done by I. P. Sharp. Much effort was put into the user interface, and into documentation and online help facilities. After successful acceptance testing, SEBPOST was put into production on September 1st, 1984. At that time, the prototype had some 270 active users, and the pilot production system about 65 users.

## **THE INTERNATIONAL BANK**

Being an international bank, S-E-Banken is building an international data communications network to allow communication between the various parts of the organization, and between S-E-Banken and its customers—several of them being large international corporations.

This 'SEBNET' combines IBM's SNA, international packet switching networks, and Digital

Equipment's DECNET, and uses a number of IBM mainframes and DEC VAX computers. It makes the company's systems and data bases a common resource throughout the Swedish and international parts of the organization. It is easily seen how electronic mail and conferencing can be used to improve communications within such widespread operations.

A link from SEBNET to the I. P. Sharp system in Toronto is also planned.

Traditionally, much international messaging uses the Telex system. As in other systems for electronic mail, a link to that network is of great interest. At present, an IMS system allows users to send Telexes via an IBM Series/1 computer. Telexes may be of two kinds:

1. Economic transactions which are in a special format, including check digits to reduce the risk of forgery
2. 'Normal' text Telexes

Incoming Telexes are printed onto paper at the central Telex offices, but a planned project will allow the Telex operators to complement an incoming Telex with a SEBPOST name-code. These Telexes will be picked up by an APL program using the AVAM interface and forwarded via SEBPOST.

A logical continuation, which is not as yet planned, would be to allow SEBPOST users to send texts to an 'outgoing Telexes' SEBPOST code, from where they would be forwarded to the Telex network.

Interconnection with other computer conferencing and electronic mail systems through standard protocols would also be very interesting. This would give both an exchange of texts with corporate customers who have their own mail systems, and an interchange of experiences with various research networks.

Possible protocols include the CCITT MHS (Message Handling Systems) which is now in the draft stage and IBMs DIA (Document Interchange Architecture), which in theory addresses some of the problems. The important feature for the user must be that such transfers are transparent, that the user does not need to know on which machine a particular user or discussion group exists. Proposed standards in this area are found in references (4) and (5).

## **PROJECTED CONFERENCE USAGE**

Under the pilot phase, discussion groups have proved very valuable on themes like 'SEBPOST implementation plan', 'SUPERPLOT experiences', 'Guidelines for 3279 colour usage', etc. For the non-EDP introduction certain usage is projected:

- Contacts with international subsidiaries.
- Each division's management group.
- Customer experiences with automated teller machines.
- Courses and seminars.
- Ordering office supplies.
- Hints on good magazine articles.
- Questions and suggestions on deposit system.
- Suggestions for the bank's internal magazine.

A list like this could have any length. Communication, and especially conferencing, is by nature difficult to predict. The central idea with the SEBPOST system is to give people an efficient means of communication, on topics of their own choice, which are relevant to them and to their work.

## **SPREADING THE WORD**

As was stated above, SEBPOST is only now being spread outside the EDP department. The goal for SEBPOST implementation is that the majority of S-E-Banken's employees should become users. Because of this, the bank's normal Training department provides SEBPOST courses. SEBPOST is just one in a large range of EDP tools in the bank, and we have an close, and growing, cooperation with the Training department.

The courses, and the everyday use of the system, need documentation. The following are available:

- A promotional pamphlet which explains what S-E-Banken expects to gain by using SEBPOST, and what each user can expect.
- A 'getting started' card which shows how to sign on, read and write texts, and lists some commands which may be of interest.
- An introductory course documentation (about 10 pages).
- A *User Guide* (about 40 pages).
- A *Reference Card* which lists all commands.

- The *SEBPOST Reference Manual* (about 150 pages).
- The online help facility.

New users can get help and advice in several ways. They can contact the person who trained them, or the SEBPOST Help Desk in the EDP Department. Normally there is also someone within the user's department who can give help. Less urgent problems can be sent to a designated SEBPOST destination.

For the time being, SEBPOST is not being actively marketed within the company. Usage increases anyway, and we want to be certain that new users get adequate support. We encourage use where people are often absent from their rooms and where geographically dispersed groups have a common interest (such as the group of banking consultants in a region). In selected cases use at management levels is encouraged. Obviously, we also encourage potential users who have easy access to terminals—in the Swedish part of S-E-Banken there are 1600 terminals for a total of 6500 employees.

## **LIMITATIONS AND PLANS**

Some additions to SEBPOST are planned:

- Provide IMS and TSO users with a message indicating how many incoming texts they have in SEBPOST.
- Allow for the organizer of a discussion group to set search keys for texts in a group archive and to remove obsolete texts from the group archive.
- Notebooks: private and group texts for updateable items.
- Protected groups: those who are not members should have no way to find out about its existence.
- Write protected groups: only the group leader may send messages to the group.
- Secretary function: a busy person should be able to delegate reading and answering of incoming mail to his secretary, who forwards only selected texts.
- Work Queue support: assume a group 'SEBPOST Questions'. The user in the group who first handles an incoming text should be able to mark it as 'his', and others in the group who read the text should be able to see this 'responsibility' mark.

- Recapitulating forwards: it is today only possible to recapitulate a comment chain to earlier texts, not to retrieve comments made to a certain text.
- The possibility to include in a text, while editing it, the contents of a file within or outside the APL system. And conversely, the possibility to route a certain text to a file within or outside the APL system.

To improve certain of these items might require some lobbying, to get additions to the MAILBOX base product.

## CONCLUSION

The benefits of electronic mail are easily seen, and thoroughly experienced in the SHARP APL community. The benefits are most striking, when electronic mail is offered within an international network like SEBNET (or the I. P. Sharp network). It automates what is known today such as the telephone, paper mail, and Telex. Thus it is often delegated by decision-makers, as they today delegate letter and telephone communication.

Electronic conferencing encourages new ways of communicating, and new ways of sharing ideas, also between parties who have not previously communicated. This often implies that management themselves use the system, because they need to discover the new forms of communication. It is not easy to delegate the discovery process. The end result tends to be a more effective communication.

## References

1. Kerr, Elaine B. and Starr Roxanne Hiltz: *Computer-Mediated Communication Systems*. Academic Press, 1982.
2. Hiltz, Starr Roxanne and Murray Turoff: *The Network Nation: Human Communication via Computer*. Addison-Wesley, 1978.
3. Evans, Michael: *SEBPOST Reference Manual*. Skandinaviska Enskilda Banken paper, 1984.
4. CCITT: *DRAFT Recommendation X.400: Message Handling Systems: System Model-Service Elements*.
5. CCITT: *DRAFT Recommendation X.420: Message Handling Systems: Interpersonal Messaging User Agent Layer*.



6. Irwin, John J., David B. Allen, and Leslie H. Goldsmith: *A Message Archival and Retrieval System for MAILBOX*. I. P. Sharp Associates, 1984.
7. Goldsmith, Leslie H. *The SHARP APL Message Processing Facility*. I. P. Sharp Associates, 1980.

## Overcoming Resistance to New User Systems

Eric B. Herr, Vice-President, Planning and Development, McGraw-Hill Publications Company

Today's electronic information services are doing much more all the time and I would like to address some of the marketing challenges that we will continue to face.

The newness is increasing. The services specially support the more popular terminals with extra value-added provided by local intelligence that provides graphics, downloading and a whole host of emerging functions. The hardware industry's recent contribution to human factors progress is impressive: our information services support the 'windows' and 'mice' with a bit of extra software, not to mention the beginning of touch activated screens and the new generation of telecommunication products. Services are using all different kinds of networks: inexpensive one-way and two-way satellite, videotex, FM radio broadcasting, cable television, and local area networks.

We're only beginning to take advantage of the new software/hardware capabilities:

- User defined keys that truly simplify use
- Easier sign on and access to multiple services
- Graphics
- Interface with other feeds

If we have all of these new improvements and integrate them effectively, will there still be the familiar problems of user acceptance of new services? Definitely, yes. Let's talk about the first line of resistance: Sales. For systems that do something very new, the primary marketing hurdles are of just two enduring kinds. First and foremost is 'cultural resistance'. This is the personal and organizational inertia that is naturally created and sustained by an efficient current means of operating without your new service. The second hurdle is 'perceived competition'. Most services must compete for the customer 'current activity dollar' even if they serve a totally new function. Even very new or 'unique' services are usually immediately drawn into competition with other services which, although similar, serve very different purposes.

The ongoing challenge is to position the product firmly in the customer's estimation as a worthwhile service, and, of course, to maintain that service position over time.

EMIS is McGraw-Hill's electronic markets and information systems venture in the petroleum/chemical trading arena. An authorized user can use the service to locate new buyers

or sellers of oil or chemicals anonymously, then conduct a negotiation, and legally commit to purchase or sell a quantity of oil or chemicals. This could well be a parcel of perhaps \$80 million. EMIS provides that service globally seven days a week. To support this trading service there are information packages from McGraw-Hill: Platt's Oilgrams and Data Resources information; from outside McGraw-Hill real-time petroleum futures, and currencies feeds; and other petroleum and chemical information services—Petroleum Argus, Petroleum Intelligence Weekly, Petroflash, and others. There are interactive graphics that access McGraw-Hill's historical price data bases and also real-time market information feeds. This system provides a broad service bandwidth—trading, information, and communications.

The trading sequence begins when either a buyer or seller posts an offer. These are listed each time a user signs on to the system. If an offer appears interesting, a user can consult virtually all relevant information services and then actually do a deal.

New users can usually find additional ways of using EMIS's service every time our customer service staff visits their location. The more functions per user the more cost-justifiable the service. International EMIS users are twice as numerous as domestic users. In places like Kuala Lumpur and Adelaide, Australia, service means continuing help during and after installation—modems, software, screwdrivers, and cables, special plotter routines, additional users and functions. And service means usage dollars.

From a marketing point of view, if we look at what service representatives are really doing of greatest value it is the maintaining of the customer relationship—relationship management.

As Ben Shapiro of the Harvard Business School will tell you, relationship management is what service marketing is all about. In the online business a chart might be constructed to show points of 'equity' in the relationship with a customer.

At the time the customer first comes to know of a supplier—by reference, advertising, or some other means, points are established. A reference who says "they are a quality service" may be worth 100 points. Let's say it takes 750 points out of 1000 to get a major contract in place at a large company. For a business service, a solid first visit from a marketing representative might add another two hundred points.

After creating initial interest, the marketing rep now must sell the organization as a whole—not just the decision-maker. The buyer is part of the larger array of the interests to be considered. That can include: M.I.S., telecommunications, finance, and any operating personnel considering similar purchases. In the cost justifications that accompany consideration of your proposal, the numbers always seem to gradually show that after consideration of many additional factors, the proposition may indeed be actionable but is probably not quite as attractive as was thought.

This gradual discovery costs relationship points. For computer-related services, the various

organizational inputs all advise the prospective buyer why they personally have not used the service. The buyer hears more and more questions about the product. Cognitive dissonance sets in.

During this period the marketing organization is working hard to provide adequate answers to the questions, to continue building equity rather than allow it to be eroded. This is where most possible sales are lost. The bigger and newer the system, the longer this period may be required to continue before a major commitment. For EMIS this has been as long as one year, although the average is thankfully much less.

One of the difficult problems that comes up in a very new system is that awareness of new capabilities leads customers to question if a half-step generational improvement might not be worth waiting for. Hardware manufacturers know this lesson; as the online services become more complex, the problem becomes ours. At this point, even though the product has successfully competed against all competitors, it must now compete against products not even on the market!

Favourable feedback from initial EMIS pilot users is critical if equity is to accrue positively. The final commitment may still be two or three months distant. Meanwhile, customer support people are training users as part of a pilot or demonstration agreement. These service support activities build relationship equity points, but they cost dollars, the full extent of which is not likely to be recovered immediately.

Let's say that all these hurdles have been overcome, and the product's strengths are fully appreciated by the company and they make an oral commitment to buy. What happens?

The sale is purchased by the supplier using relationship equity points. The day the commitment is made, this buyer debits the supplier's relationship account. And once again, the burden of relationship improvement rests upon its service department.

Price increases cost relationship points, and expansion of service under current terms adds points. New technology updates over time add points, slower acceptance of newer and clearly improved technology costs points. Service that is perceived to be of high quality always can add points—even when a technology supporting the product is seen by the customer as ageing. A graceful, non-disruptive conversion from one generation of the service to a newer one adds points.

For EMIS, we have added points with the introduction of floppy-disks for autodialing, downloading, user-function keys, plotter routines and new information services.

Every business has point values such as these that are driven by the customer relationship life cycle. If an attempt is made over time to monitor these numbers, they can serve as strong predictors of how the market will respond to a new product or price. They also help

focus an information service organization on the key determinant of success or failure—the customer's perception of the service relationship.

As you might expect, getting new users comfortable quickly is the key to implementing a new system in the organization. We are getting more and more committed to user education—finding new ways to do it. One of those ways is the McGraw-Hill interactive authoring system—a system we developed for ourselves, but which we also now market. I think it has strong application to the online business. For corporate electronic information services, finding a solid approach to in-house corporate training can dramatically increase sales. This system allows fast and inexpensive preparation of tutorials. It works on the IBM PC and can also use interactive video if desired.

Lessons can be written and run immediately—without course installation or compilation. And a powerful editing facility speeds authoring along by letting trainers make changes as they write.

Combinations of text, colour and graphics, computer screens and video segments and even computer simulations can be created.

The system comes with these components:

- An Authoring System Diskette that allows you to create lessons.
- A Delivery System Diskette that allows trainees to run your lessons.

You start by inserting the Authoring System Diskette and any blank diskette into your microcomputer.

The system offers a choice of five different types of 'lesson display screens' (these are the screens the trainee will actually view):

1. Presentation screens
2. Multiple-choice screens
3. Matching screens
4. Fill-in-the-blank screens
5. Application/simulation screens

By answering simple questions and making selections, you define the type of screen you wish to build, the location and placement of each screen in the lesson, and how one screen relates to another screen in the lesson.

If you want to individualize learning, you can instruct the system to evaluate the trainee's response and move to another part of the lesson based on the response.

If you want to integrate videotape segments into the lesson, you can do it easily and efficiently. Simply take any videotape and, using the system, define the segment you wish to include in the lesson. The system will treat your video segment like a lesson display screen.

You can easily edit the lesson on screen as it's being created or while you are running the lesson. After you've completed the lesson, you can also reproduce it.

To run the lesson you've created, the trainee uses the Delivery System Diskette and the diskette containing the lesson you have created. That's it.

Whatever your approach to user training, every dollar you spend will be well rewarded. Figuring out how best to spend those dollars is hard.

Cultural resistance is a continuing challenge...and a barrier to entry for your competition if you overcome it more quickly. And the process is ongoing. For new systems, service feedback identifies key cultural resistance points—a complicated routine, for example, which if overcome can allow the product to more quickly become the success it deserves to be.

A key part of this ongoing development is the pursuit of the Holy Grail—user-friendliness. For most of us user-friendliness is a little like painting the house; even though we do major work every few years, the house can always use a touch-up. How do we decide what to do? What to pay for? I would like to share with you some cautionary insights. Let's start with a definition:

User-friendly systems permit the user to solve a problem reliably and accurately with perceptibly less time and expense than the alternative solutions.

Consider the implications of this definition: First, the superficial user interface is only one factor in user-friendliness, and even a very complex command language requiring substantial training may be very user-friendly in some contexts. This is the computer literacy approach to user-friendliness: train the users.

Secondly, user-friendliness is relative to the group of users and to the alternatives; no software can be user-friendly across the board.

Thirdly, user-friendly relates to solutions, and not to the tools themselves. Together these mean that no system can be user-friendly except in the context of specific problems for specific users.

Finally, a system which is easy-to-learn need not be easy-to-use, and every user-friendly

system faces the tradeoff between these two goals. VisiCalc is successful because it is easy-to-use and not because it is particularly easy-to-learn. Importantly, being user friendly for one application may be counter to being user-friendly for another.

Essentially, a system is user-friendly if it solves the problems of the user reliably. (This is an admittedly less comprehensive definition than above.) Reliably in turn means that the probability that the solution will be found ( $F$ ) is above some (high) threshold. With little loss of generality,  $F$  can be thought of as being the result of three factors:

$P_0$  — the probability the user will find the set of steps to solve the problem.

$p$  — the probability the user can successfully execute each step.

$n$  — the minimum number of steps in the solution.

$F = P_0 p^n$  and the requirement is the  $F$  be at or above some threshold value  $F_0$  determined by the characteristics of the alternative systems. All three characteristics are part of ease-of-learning and ease-of-use.

The ideal system has both  $p_0$  and  $p$  as close to one as possible and  $n$  as small as possible (it obviously has one as a lower bound). Unfortunately, we can't just choose these characteristics.

Primarily, this is because generally  $p_0$ , the user's ability to find all steps has to fall as  $n$ , the number steps required increase while  $p$  generally increases as each step is made smaller. The classic easy-to-learn system steps the user through hierarchical menus. As long as the user can easily identify where to begin,  $F$  will be very high for problems solved by that system. The weakness of such a system is that it is inherently limited to the problems selected by the menu builders. Even if other problems may be solved by the system, if they are explicitly listed in the menus, their  $F$ -score is likely to be very low.

**Any general system designed for a mass market can be user-friendly only for relatively simple problems.** If the problems to be solved are not simple, it is unlikely that any general system designed for a mass market can be user-friendly.

But, if by training and documentation you can raise the computer sophistication of a group, you also raise greatly the scope and complexity of problems for which the system is user-friendly. Alternatively, by careful specification and customization to the needs of a particular user group, you can build a user-friendly system for them even if their computer sophistication is not exceptionally high.

Are integrated systems such as Lisa and VISION user-friendly? Obviously they are for certain user groups and certain problems, but...limitations exist:

1. There are limits to spoon-feeding. Lisa and VISION succeed exceptionally well in packaging their capabilities. This raises p very, very high but also to raises n by generally requiring multiple steps to do anything. These systems are user-friendly much as Tinker Toys are. They provide very easy-to-use building blocks, but you cannot so easily create large, stable applications with them. This is fine because the goal of these systems is to let more people use personal computers and not necessarily expand the scope of problems solvable on personal computers.
2. Technology poorly addresses telling the user what to do as opposed to how to do it. Lisa and VISION make spreadsheets, graphics, word processing and similar tools easier to use. The user is guided step-by-step through using this functionality. Unfortunately, the underlying premise is that the user knows what data to obtain and from where, what transformations should be made to it, what other processing is required, what report format presents the desired information and what graphics should be used. In the real world, the user may require far more assistance with these issues than with how to use the software. This is the problem 'expert systems' and 'artificial intelligence' seek to address.
3. Much of the business workday is spent on routine as opposed to ad hoc tasks. Yet, routinization is almost the antithesis of the integrated systems approach. Lisa and VISION may make each step of doing a daily report easy, but the user still has to remember and execute each step individually every day. He or she would far rather select, touch or check one choice, supply any parameters then let the computer step through the various tasks.
4. Any integrated computer system is still but a component of the total business system used by any professional. People like to take work home—at least some of us do. Integrated workstations can create as many problems as they solve by providing the user with so much power but in an isolated environment. Local area networks and, probably more importantly, lap-sized computers will help reduce this isolation. On the other hand, the difficulty of merging into the integrated applications other, existing applications will hinder the acceptance of these products.

We can't assume general computer literacy if we want to sell to all the new business users. And experienced users have ever-escalating expectations. Maybe we have two literacy levels in support software. So go carefully on these crash programs to support your services with the new software and hardware. And continuously reassess how you might better educate users. The problems are real. And we are all working on them. It will take more effort and more new ideas if we intend to tap the potential of cultural and knowledge gaps represented by the wave of PC's sweeping into business organizations. Let's just remember we are all out surfing because we believe there are big waves. Especially with big waves, catching the wave and riding it successfully to the beach is the challenge—not the other surfers.



# **Managing and Implementing International Systems: The Human Factor**

Session leader: W. Lee Bettes, Director, Banking Products, I. P. Sharp Associates

Speakers: Sheila E. Goldman, Systems Officer, The Northern Trust  
Paul M. Blake, Multi-National Systems Manager, Xerox Corporation

## **Abstract**

International systems are inherently more complex and difficult to install than domestic systems. The technical problems related to international systems have been analysed extensively, however, the human issues have generally been ignored.

The consequences of ignoring the human issues involved in international systems can be severe, since the success or failure of any system is ultimately determined by its end users.

This paper will focus on the interpersonal and human issues related to the installation of international systems. It will discuss ways to accurately identify end users and communicate effectively to them, and suggest how to respond to common problems that may significantly affect the success of an international system.



Throughout the 1970s, many large corporations installed domestic computer systems. By 1980, the primary domestic systems were complete and corporations began to consider using international systems to integrate home offices and foreign branches. The present emphasis is on implementing and installing large international systems, and is likely to continue.

International systems are inherently more difficult to install and manage than domestic systems. Although the basic technology may be similar, introducing different cultures, languages, and time zones to the installation and management process complicate the project significantly.

In recent years, various aspects of international systems have been the topic of journals and magazines. Literature on the subject discusses methods and technology endlessly. However, it universally omits the essential human factor. The home office and branch office employees must help in the implementation phase and are the actual users of the system once it is installed. These are the people who ultimately determine the success of an international system; if they do not accept and use the system, then it is a failure, regardless of its technical merits.

This paper will leave the technological and methodological discussions of international systems to others, and focus on why people are a crucial factor, how to identify those people who are important in the success of a system, and suggest some approaches for dealing with multiple levels of users and their unique needs and concerns.

## **WHAT IS AN INTERNATIONAL SYSTEM?**

The most common definition of the phrase 'international system' is:

*A system that is used by people in two or more countries.*

As defined by the home office, such a system will normally include a central computer, a communications network, and end users in multiple locations. This definition is illustrated below.

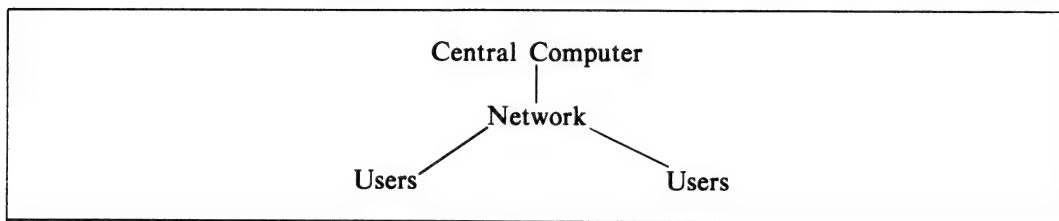


Figure 1

Unfortunately, this diagram ignores the perspective of those people who the home office expects to actually use the new international system. To the branch office user, an international system might be more appropriately illustrated by the following diagram:

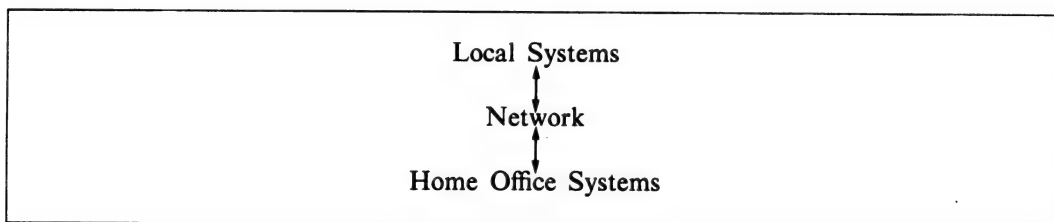


Figure 2

In Figure 2, considerably more emphasis is placed on local systems. Home office systems are present, but they, like taxes, are often perceived as simply a requirement of doing business. Since branch office users are more concerned with making their own systems succeed than with "feeding" a new system mandated by the home office, the phrase "end user" in Figure 1 is better represented by the phrase "local systems" in the perception of local users.

A more universal definition of 'international system' might be

*A system that facilitates the flow of information through an organization with employees in at least two countries.*

A graphic illustration of this definition is:

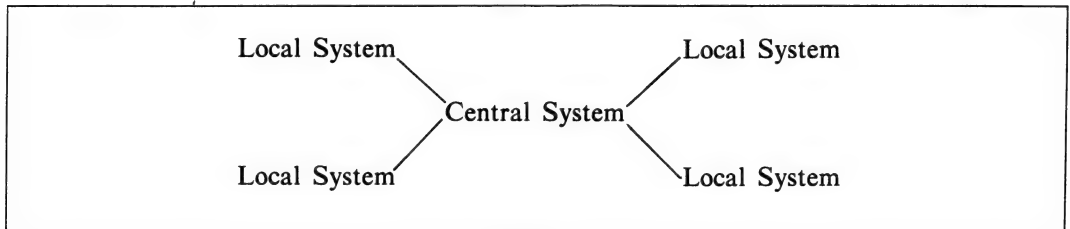


Figure 3

In managing and implementing international systems, understanding this definition is important because it implies at least five different end users: a central user and four unique local systems with multiple users. The users of each local system have different specific requirements for an international system that are, to a large extent, defined by the particular local system. The diagram intentionally omits the network, since end users are generally unconcerned with it.

The phrase 'facilitate the flow of information' is also essential in defining an international system. It introduces the concept of reciprocal information and benefits between the home office and the foreign branches. Local users will only be willing to support and participate in an international system initiated by the home office if information and benefits are perceived by both groups.

### **WHO ARE THE USERS?**

An international system, even if well-designed and properly installed, will not succeed if it does not address the needs and concerns of the end users. Anyone who has been involved in installing an international system can tell horror stories of how each foreign subsidiary disrupted an orderly installation in one way or another and prevented the system from ever being installed. There is one way to avoid this situation: carefully identify the end users of the international system and communicate with them.

Although this sounds simple, it is complicated by the fact that every international system has different levels and sets of users.

From the perspective of the head office, a company usually has

1. a head office, and
2. foreign subsidiaries

These are connected by an organizational structure that resembles the diagram below

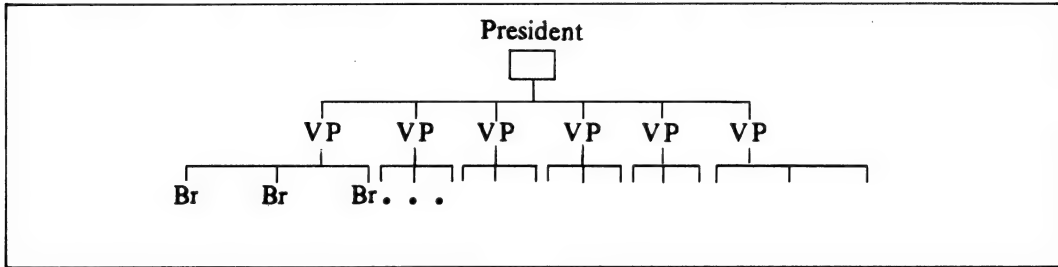


Figure 4

This chart clarifies decision-making and reporting relationships. However, people who are attempting to implement international systems cannot afford to accept such deceptively simple representations.

Based on Figure 4, users will be designated by the management hierarchy. However, the actual nuts and bolts director of a local system may not even appear on the chart. The result of accepting this diagram is that the management-designated user (usually the senior branch official) becomes an unnecessary communication filter between the international system project manager and the real local users.

A more accurate organizational chart might look like this:

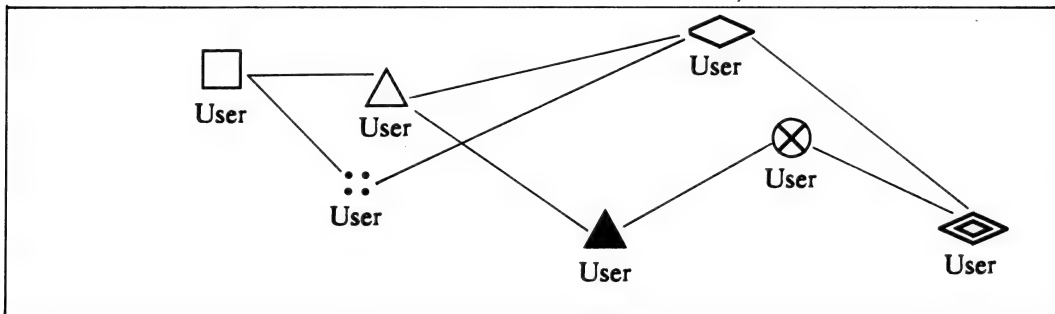


Figure 5

This diagram represents the concept discussed earlier: each sub-unit is unique and has little, if any, connection to other units. Each person who will be involved in an international system perceives his/her role and needs as central. This perception cannot be ignored if an international system is to succeed.

At the very least, it is essential to ask enough questions to find the director of the local system at each foreign branch. Next, talk with someone (preferably several people) from every department, unit, or level that might conceivably be involved in the system at some point. Do not be diverted by advice that 'you don't need to worry about XYZ department or the London branch...' If possible, travel to each foreign branch and meet the management and systems people.

Then, publish and circulate a list of all potential users of the international system. This list represents a public acknowledgement that each potential user is important. The people on the list become an informal 'system steering committee'. The committee should be kept informed of developments as the project progresses. It is worth the effort to establish a working relationship with the steering committee.

## **COMMON PROBLEMS**

### **I'm from Home Office and I'm Here to Help**

This may be the most often-repeated lie in the world of international systems. As an international system is designed and installed, the manager should be working closely with representatives of each foreign branch. Begin with the assumption that the foreign branches have a less than complimentary view of the home office. Understandably, the branches may perceive the international system project manager as an emissary from the home office and receive him/her with considerable suspicion.

It is the project manager's responsibility to gauge the level of mistrust and disprove the branch's general opinion of the home office. There are two simple steps in achieving this: first, find out what the branch users want, and second, ensure that the international system provides it. Also, try to present the benefits of the new system in terms of the benefits it provides for each particular branch.

Additionally, a project manager from home office may have little or no credibility overseas. Credibility must be deliberately established each time that the project manager deals with a new group of people. If the corporate culture is fairly pervasive, a project manager's credibility may be enhanced by identifying the new system with a common corporate goal. The system can then be presented to local branches as an implementation of the common objectives shared by both local branches and home office.

This is where the project manager's relationships with local branch personnel are tested. Communication as to the needs, concerns, and preferences of local users is essential at the development stage of the system.

### **The 'Not-Invented-Here' Syndrome**

Foreign branches are usually satisfied with their local system. Generally, they do not feel that interference from headquarters is necessary, and certainly not desirable. The suspicion and mistrust that are characteristic of the 'Not-Invented-Here' (or 'NIH') syndrome will persist until the project manager has successfully convinced each foreign branch to accept the new system.

The only way to address NIH is to listen and carefully analyse each local system in order to understand its shortcomings and problems. A tactful comparison between the local system's problems and the advantages of the new system may be very helpful. Tact and courtesy are central in dealing with any international system. Foreign branch personnel should not be put in a position of having to lose face in order to accept the new system.

### **Reverse 'Not-Invented-Here' Syndrome**

The fact that the new system was designed and is being built by home office does not guarantee that it is superior to local systems. Considerable branch resources have been invested in developing local systems that address uniquely local requirements. Conceivably, local systems may be superior to the new system in those (or other) respects. Careful analysis may indicate that the local solution is the best response to a particular problem.

Be open to this possibility and investigate bridging opportunities. Seek out interface alternatives that are least burdensome to the local branch. If feasible, allow local branches to choose between using features that are duplicated in both the local and the new system. In general, although central control is usually the main reason for a comprehensive international system, try to ensure that the new system does not significantly encroach upon the existing authority of branches.

### **English Is Our Official Language**

Although the company may have a primary language, it is foolhardy to assume that everyone in the foreign branches fully understands it. Since communication is so important in developing successful international systems, it is advisable to find out what the primary languages of the end users are and their general proficiency in the company's primary language.

In dealing with foreign branches, avoid using jargon, acronyms, and slang. Ask frequent questions to verify that you are being understood. Since people generally learn the courtesies of a language before they learn its substance, keep in mind that initial introductions may not accurately reflect language proficiency.

Cultural differences are a corollary to the language barrier. A project manager must be knowledgeable about foreign cultures and their practices. Each country has important cultural norms that a project manager is wise to understand and comply with.



There are various ways to address a language problem (using an interpreter, for example). Regardless of the method selected, the issue must be addressed. If not handled quickly, the language barrier can significantly impede the progress of the entire project.

### **We Have Our Own Way of Doing Things**

Most foreign branches installed primary computer systems not long after the home office. Unfortunately, it is likely that each foreign branch installed a different system and developed their own naming standards. An international system must mediate between different sets of naming standards and interface with all the local systems.

Different naming standards is one of the most common problems international system managers must respond to. It can be handled in various ways. The home office can publish naming standards and require that all foreign branches comply. Alternatively, a complex translation table may be built into the international system to incorporate aliases. Project managers should be aware of the existence of local naming standards, and allow extra project time to respond to the inevitable conflicts.

Although developing interfaces between the international system and various local systems is primarily a technical issue, project managers should schedule extra time to work with local branch system managers to sort out the various technologies involved in the interface.

Similarly, the availability of terminals and responsibility for their ordering, installation, and service is another potentially frustrating technical issue that may disrupt working relationships and the installation of the system. Terminal availability and service are often particularly difficult in certain international locations. In this situation, project managers should attempt to provide as much information on terminals, availability, etc., as feasible. It may be helpful to diffuse frustration by assigning a technical advisor who will be available to answer questions and act as a liaison between the international offices and the company that is providing the terminals and service. Alternatively, it may be advantageous to assign the terminal issue to one representative from each local branch. This enhances local involvement in the project; it also simplifies coordination between the technical manager and local branches.

### **APPROACHES**

There are several strategies for dealing with common problems. The majority respond well to thoughtfulness, communication, and extra time. Some specific suggestions are outlined below.

1. Expect an international system project to take about twice as long as a comparable domestic project. Schedule extra time to resolve unforeseen problems that will inevitably occur.

2. Build an international project team. Use home office people who have had international experience. Consider bringing representatives of foreign branches to the home office temporarily. This will facilitate communication with the particular branches, and foster a sense of local ownership of the international system.
3. Encourage top management to explicitly and openly support the new system. This may partially diffuse the internal conflicts that often accompany an international system.
4. Communicate. Communicate. Communicate. Hundreds of decisions are made in the course of a project. Use memos, letters, and phone calls to communicate these decisions to foreign branches as often as possible. If it is feasible to solicit their preferences on certain decisions, do so.

Ideally, communications should be simple (in recognition of language conflicts) and frequent. Although it is not uncommon to publish a large document six months after a system goes live, it is much more effective to keep everyone's attention through continuous communication.

Electronic mail is almost indispensable in dealing with international systems. It cuts across time barriers and provides a clear audit trail. Additionally, since it is easier to understand and translate written communication than verbal communication, electronic mail minimizes language conflicts.

5. Travel. As a corollary to written and verbal communication, face to face contact is very important. Although it is unlikely that the company will authorize an extensive travel budget, project managers must insist on some travel budget. It is important that the project manager establish him/herself as more than just a voice on the phone or a name on a memo. Foreign branches are more likely to support the implementation of a system when they have a working relationship with the project manager.
6. Execute a comprehensive systems acceptance plan. This should avoid the installation of a system that doesn't work, and reveal most bugs and problems. Installing a system that does not work creates so negative an impression that even full repairs may not convince local branches that the system is reliable and workable. Comprehensive systems testing, although time consuming, will address this contingency.
7. Plan for phased implementation. Although every project manager would prefer the thrill of having the system go live all over the world simultaneously, attempting to do so may be disastrous. Problems associated with different technologies, naming standards, and language barriers are easier to identify and resolve one installation at a time. As the project team becomes more adept at resolving these problems, subsequent installations will be simpler.

8. Documentation. Due to language barriers, physical distances, and time zone variations, the project team may not be accessible to answer all questions as they arise. Therefore, user documentation must be better than average. The burden is on the project manager to ensure that user documentation is comprehensive, specific, simply written, and easily understood.
9. Delegate interfaces. Delegating interfaces to local branch personnel is a helpful technique. Try to select and support a 'local expert' at each branch to coordinate the interface process and enhance communications. First, local people are most familiar with the technologies of local systems. Local expertise will simplify the development and installation of interfaces.

If it is feasible to localize support of the new system, do so. Branch offices will work more effectively with support personnel who are located in the area, speak a common language, and are familiar with local customs.

Additionally, the more that local people participate in developing and installing the international system, the more they will feel a sense of ownership towards the international system. One alternative is to establish an 'international system support request list' that all branches may review and supplement. This enhances communication, and reinforces the impression that local branch needs are of continuing concern to the central project team. Local users are more likely to support such a system.

## PROJECT TEAM

Traditionally, project teams for large international systems consist of a technical manager and a team. However, installing and managing international systems requires so much more communication and coordination that an alternative structure is recommended.

At I. P. Sharp Associates, both a project manager and a technical manager are assigned to international systems. The project manager is responsible for coordinating the broad details associated with implementing and managing the system. The majority of his/her job is developing working relationships with foreign branch employees and keeping communications flowing smoothly. The technical manager supervises the project team and is responsible for designing and developing the application code.

Additionally, the project team includes programmers (both central and local) and at least one technical writer. Since the success of the international system depends heavily on the quality of the user documentation, it is recommended that a technical writer be present for all stages of developing and installing the project.

**CONCLUSION**

The true challenge of international systems is not technical, but human. Foreign branch employees are geographically dispersed, speak different languages, and may have different (if not directly conflicting) needs for an international system. It is crucial that an international system project manager be sensitive to potential problems and meet them head on. To ignore the unusual human issues inherent in international systems is to lay the groundwork for the practical failure of a technically superior system.

# **SOME MANAGEMENT TOOLS FOR THE INFORMATION CENTRE**

William Apsit, Data Processing Manager, I. P. Sharp Associates

## **Abstract**

This paper discusses the Information Centre in terms of the problem it solves—the incompatibility between the traditional computing centre and non-traditional end users. These users are the ones requiring 'immediate response', who have 'once-off' jobs, and whose specifications often never freeze. It views the Information Centre as directly supporting these users and looks at an environment that includes not only personal computer use, but also use of a central computing centre, communications network, and non-local databases. It offers tools, broadly-defined, which an Information Centre manager may find useful in this environment.



## **INTRODUCTION**

### **The Problem**

Corporate computing centres have traditionally had trouble with one category of users and potential users, and often still do. These are the users with 'once-off' job requirements, users with a demand for 'immediate response', and users who can never freeze their job specifications. When we speak of 'traditional' difficulty here, we refer essentially to the last fifteen years. Before that time the technical level of the available hardware and software made it almost impossible for a computing centre to support such users.

The computing centre generally provided batch-processing service, spending most of its resources on very large-scale jobs that usually involved considerable record-keeping. Accounting applications and inventory were typical. The jobs were well-defined, initially replacing manual systems, and requirements for changes usually came with sufficient notice.

Now, with personal computers, these non-traditional end users are attacking their own processing needs. They look after their own once-off jobs and special analyses. More and more software packages are available to them. Sometimes, however, they have trouble getting the data they need. Sometimes their micro-computer is too slow. Sometimes they duplicate each other's efforts.

### **Origin of the Problem**

IBM began delivering its first commercial mainframe in 1955. The IBM 702 ran one job at a time, and was slow and unreliable by our current standards. It was expensive and few people were users. The applications were largely accounting and inventory.

The second generation of computers provided better price-performance and reliability, and as a result became the first computer system in most businesses. Service bureaus thrived, some providing packaged software. Applications broadened, and came to include things like classroom scheduling as well as accounts payable and payroll.

Most current computing centres descend from these early generations of commercial mainframes. Computing centres have grown enormously from this time. Now desktop personal computers provide more computing power than typical mainframes of fifteen years ago. The fact that today's personal computers do not require similar staffing and environmental needs does not mean, however, that the early data centres were inefficient. It simply emphasizes the considerable improvements in computer technology.

The small models of IBM's third generation, which began to appear in 1965, had similar computing ability to a personal computer today. They were the first generation to support

multiple programs, significant data communications, and removable disk packs. They were a tremendous improvement in price-performance. And they enabled many users to justify the feasibility of their applications. With them came the nucleus of the information centre users we have today.

In 1965 and for a few more years these users were still out of luck. The computing centres, given the vast improvement in hardware price-performance, developed considerable backlogs of useful work. Users who had not previously used computer centre services often lacked the contacts or clout to get themselves into the queue. Some of these users took their applications to service bureaus who wanted their business. The value of timeliness often forced them to move.

I.P. Sharp was one of these service bureaus. It offered an interactive timesharing system supporting the APL language. It gave three-day courses to make users proficient enough with the language to program their own applications. And it offered consulting and programming services to design and program systems for customers on a project basis. Responding to 'once-off' applications, the need for quick interactive response, and providing instant support with a phone call became our bread and butter. This responsiveness and sensitivity to user needs still serves us very well.

Corporations, however, became upset at the large number of outside firms providing their departments with service. When they looked closely, they also found duplications of effort, duplicate (or only nearly duplicate) files in different locations. Conversion programs were often needed to move data from one location to another. Overall the situation was very complex, and within the corporation the only entity in touch with all of it might be the accounts payable department.

**Example:** About ten years ago a major U.S. bank looked at its use of outside service bureaus. It learned to its surprise that it was using eight of them, although it had a very large computing centre itself. Being upset by the unexpected proliferation of outside services, it engaged in a program to cut back the number from eight to three. In the process it gained valuable insight into the kinds of work various end users were doing elsewhere. And it determined to do most of this work itself, and dedicated a large mainframe to this work. It found duplicated software, duplicated data, and incompatibilities galore. Among other things it found corporate data which it considered highly sensitive in unapproved locations outside its control.

## **A Solution to the Problem**

It is fundamental to recognize that use of personal computers gives better price-performance for raw computing than a corporate computing centre or a service bureau. And it's fundamental to recognize that this will continue to be the case.



Recognizing the importance of access to data is also essential. But here a central control is a logical one. Much of the data a user wants exists in the corporation somewhere, and it's certainly good if a user knows who to contact to learn about it. It's even better if this contact is familiar with useful databases publically available outside the corporation.

Centralized knowledge helps avoid a duplication of effort. It could help avoid repetition of experiences with poor hardware or software. It could introduce users with similar requirements to one another. It could reinforce requests for service. It could obtain better pricing for larger-volume purchases.

The user whose microcomputer is too slow for some purposes could benefit from service from the large mainframe computer. Most users could benefit from electronic mail, which would also be available through the mainframe. A considerable wealth of corporate software on the mainframe, as well as data, could benefit these users.

So, of course, we have the Information Centre. It exists as the primary support vehicle for these end users. In certain cases it might do a bit of programming or refer specific systems to the corporate computer centre for implementation. But for the most part it consults with these users, provides them with education and support, and suggests appropriate hardware and software. It responds.

**Example:** We hear again and again about the considerable backlog of development projects in a traditional computing centre. L.W. Hammond reported a case in which an organization looked at the backlog from the point of view of which parts were suitable for an Information Centre. The projects were apparently on the backlog because they were already justified in terms of implementation and running costs and user benefits. Estimates said that the portion of the backlog amenable to the Information Centre approach could be done for 1/3 to 1/5 the cost that was already justified. A pilot effort confirmed those estimates and the Information Centre became permanent. [Hammond, p. 142]

### **The Future of the Information Centre**

Over time, further price-performance improvements in the microcomputer world will make more and more applications feasible. More and more of the current and future users will require access to central data bases, both for corporate data and for general economic data. The Information Centre will track technical developments and the sources of data. It will also look at costs and capacity planning.

It is important to recognize that the increase in microcomputing will not result in a decreased demand for central computer services. On the contrary, there will be considerably more users in all, considerable quantities of shared data, and increased communications. An acceleration of mainframe growth is likely with the enormous increase in local processing.

In time it is likely that more than half of a corporation's computer hardware expense will be outside the computer centre itself. And it is likely that the bulk of this will be coordinated and looked after by the Information Centre. The coordination will include projections of required processing, storage, and network use for the computer centre.

In 1990 most of the end users supported by the Information Centre will be ones who do not currently make any direct use of computing services. At this time the Information Centre users in a typical large corporation will probably be using central site resources that exceed the size of a typical large mainframe now.

**Example:** this trend is already apparent in the internal Information Center of IBM Canada today. In the past eighteen months the number of mainframes dedicated to end user support has increased from four to eighteen. In the first three quarters of 1984 an additional 250 personal computers went to Information Centre users. The company itself has somewhat over 10,000 employees in Canada. Considerable further growth is likely.

## GENERAL MANAGEMENT

Many management provisions and techniques are common, whether in an information centre or an oatmeal factory. In many endeavors being responsive to end users is essential.

In general, a manager is one who gets work done through other people. The ability to work through people is fundamentally important. It's taken for granted that the work a manager is responsible for is considerably more than the manager could do alone. The manager is basically responsible, as well, for the allocation of resources and efficient use of resources.

**Example:** By the early 1100's Venice had become a powerful military sea power and trading power. Nevertheless she feared for her future, and her Doge, Ordelafo, inaugurated a vast shipbuilding plan. He replaced the small private shipbuilders scattered around the city with a large central industry. It included docks, shops for the trades, foundries, and ammunition stores. For faster assembly and repair it used interchangeable parts. For ease in identifying them, they were given numbers. Venice developed different ships for war and trade, though they included enough cargo space in war vessels to enable them to be economically self-supporting from their trade. And they included defense capability in cargo ships. They became able to produce warships at the rate of a few a day on short notice. They could do major refits quickly. When the mariners' compass was available, by 1275, they used it to produce better charts of the seas. When the rudder became available, it was no longer necessary to steer by placing an oar in the water to one side. They made use of the rudder to sail year-round, even in winter, and to produce larger ships (since there had been a much lower limit on the size of a ship which an oar could turn). When it was economically justified to move cargo more quickly by adding,

say, two hundred oarsmen, they did so. And they let the oarsmen take some goods for personal trade and profit, without charge, to provide a further incentive to them, beyond their salaries.

So we have an early example of benefit from centralization, standardization, efficiency, specialization, contingency planning, and adaptability. All of these are well recognized as useful today, though not popular in the 1100s and 1200s. In a sense one can think of the city, Venice, as a corporation. Its arsenal, from which the word was coined, developed to serve its traders, its 'users'. It met their requirements for timeliness, security, and efficiency. In a sense this 'corporation' was a servicer. Many of these ideas and techniques were very creative for the twelfth century. They apply well to the Information Centre today.

## **INFORMATION CENTRE MANAGEMENT**

### **Objectives**

Planning is especially concerned with preparation for events or situations that can be expected in the future. Sometimes we plan for things which we do not really expect, but which we can nevertheless imagine, and for which we determine some probability.

In an Information Centre or a Computing Centre one of the objectives is inevitably a desired level of service. This is often measurable in terms of response time to trivial requests, number of failures, mean time to fail, and failure duration. We also look at response time for transactions, problem resolution, and user system development.

System availability is also an important service level goal. Often it is quite possible to increase system availability, measured as the percentage of scheduled hours during which a system is indeed available to users. Analysis of the cause of previous failures helps enable this. This analysis might reveal, for example, that each unit of a particular brand of disk-drive fails after six months use.

The manager, however, is usually faced with trade-offs or possibly unacceptable cost-benefit computations. Installing no new system software in 1985, for example, might improve uptime by 0.1%. However, the cost advantages of the new software probably make this an unacceptable trade-off. Use of an Uninterruptable Power System, projected to save two hours a year of available time at a cost of two million dollars might lose out for cost-benefit reasons.

**Example:** It's almost a classic case that response time for trivial system use and transaction turnaround times will improve significantly when additional disk controllers serve certain data paths. The cost of the additional controllers is known precisely and the improvements from installing them are often estimable with great accuracy. Although both cost and benefit are known, however, the decision about whether

or not to install the additional equipment is often not an easy one. In many instances the cost in such a situation, by virtue of being avoidable, is avoided. Nevertheless, recent studies indicate that a system with extremely prompt response has considerable economic benefit to its users.\* An added filip to the installation of additional disk controllers for the sake of improved response is that the additions can also provide protection against the failure of existing controllers. This ability to serve as backup not only continues to make data available in the event of a controller failure. In I.P. Sharp Associates' experience, it also helps reassure users and system auditors about the priorities in the services provided.

An important point about objectives is that if we define them precisely we can measure our attainment of them. We can check, at the end of a year, whether the incidence of water-supply problems that caused system outages changed when we arranged to inspect the system three times a day instead of one. Further, if we know that the cost to us of an outage is \$10,000 plus \$25,000 per hour, we can make those cost-benefit judgements quite readily. In a case of this sort, our estimate of costs does not need to be precise. In many instances the anticipated benefit so vastly exceeds the cost that multiplying the cost by a factor of five won't change a decision to proceed. The converse is also often true.

## Functions

**Staffing:** We sometimes learn, sooner or later, that many of our problems are human rather than technical. This seems to be at least as true in computing-related affairs as any other. The attitudes of people are elemental in avoiding confusion, accepting and assisting with positive changes, and in understanding the problems of users.

There are a number of general characteristics that are very important in Information Centre staff. The first is quite certainly fine communication skills. Without the ability to communicate well with other workers, users, and Information Centre staff, no employee can do a very satisfactory job. Unlike knowledge of a software package, communications skills may be very difficult to teach.

We also want the staff to be self-motivated and to make judgements reasonably well. In most cases the best means of determining these characteristics is probably to judge from the person's past. Someone who has worked at three different jobs in the past two years and who found quickly that each was not as interesting as expected maybe making another judgment mistake in applying for your job opening. If the past three employers all say the applicant did fine work but needed close supervision, be warned away.

Another very positive characteristic is problem-solving skills. It's nice if patience and tact go along with this, and persistence.

The same is true with teaching ability. Most Information Centre staff will be involved with

explaining things to users and other staff members on a continuing basis. You might measure an applicant's teaching ability if the resume indicated an area of experience you didn't have yourself. Ask the applicant to explain this experience to you during an interview. It might also reveal something about the person's patience and persistence.

Previous experience may be only of secondary importance in selecting staff. People with no experience should be reasonably fast learners, of course. But the package you support today may be replaced by a different one in the future. Specific application knowledge can help understand the problems of a user in that specific area—but it may not help if the person with experience isn't a good listener. Or if the experienced person makes too many assumptions about a user's problem without verifying them. In many situations, the staff need not understand an application in great detail. After all, the goal of the staff is to enable the user to be self-sufficient.

A university background might nevertheless be a positive factor in an applicant. It demonstrates the persistence and usually some degree of self-motivation. In some cases it also demonstrates the ability to absorb a considerable body of knowledge in a modest time. For sheer persistence and self-motivation, though, my favourites are marathon runners.

Lastly, think of your Information Centre in terms of quality. Your staff reflect your department. Your users will associate any mediocrity in one of your employees with more than just that person. High quality of Information Centre staff will amplify the value of the service to the end users.

**System Availability and Performance Measurement:** An Information Centre or Computing Centre often provides interactive services to its users. System availability in such a system is critical. If the system is not available as scheduled, users suffer. The users also see system performance. If they find the system response time too slow, they also suffer. Users express strong feelings about these things.

**Example:** Some years ago I.P. Sharp Associates began keeping details about system availability online for its users to look at. Most users don't look at it very often. Many have never looked at it. However, some users, to whom availability is very important, look at it every month. Making the details publically available saves the trouble of responding to user requests every month for the information. The online detail usually tells them all they want to know. I.P. Sharp has also found it useful in discussing equipment or software failures with suppliers. Suppliers are sometimes more concerned and responsive when they know their failures would be widely disseminated and attributed to them.

IBM makes a commitment to its internal Information Centre users for both system availability and trivial response time, and reports its results regularly. In this case a user upset with response time can quickly also verify whether or not the actual experience meets the

commitment level. IBM goes further and regularly solicits feedback from its end users on these and other concerns.

Usage histograms are a useful tool to look at performance. Generally, measurements at regular intervals, perhaps fifteen minutes, work well. They record such things as average response time, tenth percentile response time, and 90th percentile response, and various queue sizes in the systems at these times. With such histograms it's possible to measure the results of system changes, and to estimate the results of future growth and changes.

**Usage Monitoring and Chargeback and Feedback:** An Information Centre costs money; presumably it exists because it is cost-effective. One of the points about something being cost-effective is that it should be measurably so. It always is useful if such things are seen and understood. If an Information Centre charges its users, the users will place a value on the services provided. And they will tend to restrain their requests to cost-justified ones. Chargebacks are also more likely to generate negative feedback from users about poor service. When something apparently costs a user nothing, the user also has less incentive to complain about its inadequacy. The feedback is essential and charges help produce it.

Data on resource charges can themselves be a form of feedback. They may provide indications of how the use of a particular resource is changing. This in turn may help in future planning. It's important that a need for additional resources is apparent quickly enough so that new equipment is on order before its lack causes great pain.

Central to user monitoring is knowledge of who the users are. One mechanism is a registration of users who use the Information Centre with followup on the information regularly, perhaps every six months. Chargeback to users also helps keep their usage statistics prominent.

An inventory of users, complete with a list of their personal hardware, software, and data used, is invaluable. Without it, it is difficult to grasp the overall scope of any of these items. Getting it and maintaining it is sometimes a problem. In most cases, an Information Centre is established only after various users have already begun their activity, perhaps independently. It is possible, though, to ferret out these users and include them in the inventory and planning.

Sometimes this can be done using an inventory of equipment maintained by the Office Services Department. Sometimes a check of insurance in place will list significant hardware systems. (Microcomputers, fortunately or not, sometimes don't cost enough to be included in such lists.) Sometimes these users subscribe to outside computing services, and a check through Accounts Payable will uncover them. I.P. Sharp Associates once had an Accounts Payable Department using its timesharing services simply to track all of their corporate use of outside service bureaus.

The most attractive way to locate users, however, is probably by offering them benefits to set registered. This would include the provision of newsletters and announcements, courses, and perhaps software discounts and invitations to demonstrations and user meetings.

Sooner or later many, if not most, of the personal computer users will want access to available corporate information or data. If they know of the existence of the Information Centre, and they understand that access to such information comes only through this entity, these users will contact the Information Centre.

Positive feedback on Information Centre services is also needed. Those of us in a computing environment often get too little of it. It's not just that we feel better if we know that we're needed and useful. It's that we do more of the things that our users apparently get great benefits from. It lets us better recognize the success, if there is some, of remedial work to improve procedures or ease of use.

**Example:** Questionnaires sometimes elicit useful responses from users. Online questions at signoff time on an interactive system are especially useful. They provide an immediacy of feedback that is otherwise very difficult to obtain. IBM Canada has successfully made use of a system which chooses a random segment of users for feedback at signoff time. Thus most users are not asked for it after most system signoffs, and most users are responsive when asked.

### **Economic Justification of Usage/Hardware/Software:**

**Example:** An impressive method for justifying usage or acquisition of resources economically, is that of IBM Canada. Each user, with some assistance from the Information Centre, must form a business case for the resource use or acquisition. To this extent, the IBM Canada situation is not very unusual. It's most interesting aspect, however, is the implication of this business case.

If a user department presented a case for saving one clerical staff person as a result of the acquisition, the user department head count would be reduced by one. Likewise if the user indicates a saving of twenty thousand dollars a year as a result, then that money will be subtracted from the user's budget. This doesn't mean, of course, that a clerical person might be laid off as a result, but that someone will transfer from one location or job to another.

This technique, setting the projected savings against the user's actual staff or expenses allowed, makes a very important point. In many other cases the savings, as defined, are often artificial and unmeasurable.

It's useful to remember, though, in any economic justification for personal computer acquisition, to also specify non-tangible benefits. They are often considerable, and in some cases

they are the most important benefits. Higher quality is sometimes hard to measure in dollar terms. The same with improved market share. Even a rough estimate of their cost impact, in extremely conservative terms, could nevertheless be very helpful in making an evaluation.

**Equipment and Software Changes:** One of the things that an initial inventory of existing hardware and software in the hands of users will reveal is that lots of the pieces are not compatible with one another. In some cases one user will already have serious incompatibility within his/her own array of resources. This is probably inevitable but it isn't at all desirable.

**Example:** A consultant reported on a collection of four packages offered by a particular vendor as a suite for personal computer management of financial data. Two different companies, the consultant found, wrote pieces of the packages. Two of the four packages would not run under the PC-DOS operating system that came with the computer. Running them required shutting off and restarting the computer. None of their files could be accessed by any standard means outside of the packages themselves. One of the packages would not start up on a personal computer with the hard disk option. [Seymour]

What do we do about the lack of standardization? We might follow the example of the Doge of Venice in the early 1100s, and begin to standardize. We might try to maximize, whenever reasonable, the interchangeability of parts, whether hardware, software, or data. In the future we will become much more productive and efficient if we do.

How far do we go with standardization? As far as we reasonably can, of course. Standardization enables our personal computers to communicate with one another. This is critical to moving data from one computer to another. Fortunately, it looks as though this kind of compatibility may not be very far away. Many personal computers take an RS232 communications interface as standard. Fortunately it was largely a standard before personal computers became available.

We also want to be able to move data between the personal computer and the mainframe computer. Often data needed for local processing by a user with a personal computer will be available in the mainframe. Often the personal computer user will want to transmit data to the mainframe for the use of others, or simply to report it to others. This can generally be done, with some planning, by a personal computer user who has a personal computer that can be a terminal on a mainframe.

Standardization should be part of the integration of facilities itself. In an ideal environment, a user whose personal computer is a bit slow for some task will sign on to a mainframe and run the software there. One central piece of software should make all data available to a user, from whatever source. This could mean combining local data on the personal computer with data from various sources, including perhaps the company mainframe and



an external mainframe offering public data bases. It's data, after all, that forms the basis for all of the reporting and analysis that the user is doing.

Sometimes it will be desirable to upgrade or replace user hardware or software, both local to the user or in the central computing centre. These changes will affect Information Centre users. 'Sometimes' doesn't mean 'maybe one day'. It means that 'we can be certain that at some time' we will make such changes. Standardization is important in such a situation. In these cases, though, our fundamental goal should be to protect the users from inconvenience.

Let's assume a software change to a standard operating system shared by a large group of end users. In principle, the change should not be incompatible with currently running systems. In practice, we don't dare simply distribute the new software and let users find out. If it's not compatible with the old system, some users will have their systems fail. They might also corrupt their systems in some way that may not be evident until long afterward. In most cases the change will probably be desirable for most users. It might correct bugs, improve the efficiency of the system, and offer enhanced function. So how should we get it safely delivered?

We might begin by notifying users that it exists and verifying from their feedback that it does look desirable. A newsletter might accomplish this. An online news system with an opportunity for user response could do the same. We then have a responsibility to test the new software. This would be an internal test in the Information Centre. If the new software works well in the Information Centre, the next test would be with a selected subgroup of users.

Ideally this subgroup would not be simply selected by the Information Centre, but would also be self-selected for such purposes. Many users are in fact very pleased to assist with such things. This subgroup should be self-contained, so that a bug it might experience would not be migrate to other users who are not part of the test. Similar subgroups might be used for testing new products which looked good after an initial Information Centre evaluation and test.

In the end, when the software is cleared, general distribution takes place. Users should be made aware of the new software and install it in a reasonably short period of time. If needed, the Information Centre would provide some user hand-holding. In any case, the Information Centre would update its inventory to reflect the change. In each case, the records should show both the delivery and install dates of the change.

This implies, practically speaking, a somewhat messy clerical task. Where confirmation of delivery or installation did not yet exist, someone must contact the user. This means that on a regular basis, in an Information Centre with many users, someone from the Information Centre would be contacting those users for whom the records are not complete. This would

probably take place every week. The calls could also solicit information about other user changes or concerns.

**Security and Backup:** Every file of data, every personal computer, and every software package should have a designated owner. There should also be a list of its users. And the Information Centre should maintain this information or have access to it and the ability to update it as needed.

It should also be a requirement for end users desiring data or a new software package or new hardware to direct their requirements through the Information Centre. In these situations the Information Centre would sometimes just act as funnel. It could simply coordinate things by passing the request for access to certain data to the individual who owned that data. But even in such a simple case the Information Centre would also act as something of a filter. It might, for example, direct a user to a different source of data which might already include some consolidation the user would otherwise simply duplicate.

In some cases the Information Centre would itself control the access to various data files on the corporate mainframe. In other cases the Information Centre would know the source of publically available data on an outside service and arrange to acquire this data. As the focal point for the requests, the Information Centre maintains the inventory of which users access which data.

The Information Centre would also often be aware of the confidentiality or sensitivity or privacy of information or software. It could ensure that users were aware of these implications and accepted the care and perhaps additional overhead required. It could verify from time to time that backup, if required, was done. It could remove access to sensitive data on a regular basis, perhaps every six months, for a reinstatement of access if justified.

In every case these controls are easier, rather than more difficult, in an integrated environment. This is true, at least, assuming that the integration includes a regular requirement for access to the company computer centre. This is one example in which a more complex environment makes for easier administration.

**Example:** I.P. Sharp Associates maintains a VIOLOG for its time-sharing system. This is a log of online information with one record describing every unsuccessful attempt to connect to the system or access data on the system. Each record includes the time-of-day and date, user account number, communications node and line number, and other detail. A large number of unseccessful attempts in a short period of time will print a message to a system operator for immediate followup by management. Smaller numbers of attempts that fail would generate automatic electronic mail messages to a local manager on the following day.

Information Centre managers agree that the Information Centre exists to do what best supports the users in solving their problems. The users are the ones who determine the data they wish to use and the manner in which they wish to massage it, look at it, and report it. If we help them to do that more quickly, more efficiently, more comfortably and easily, we are serving our purpose well.

### RECOMMENDED READINGS

L.W. Hammond, "Management considerations for an Information Center," *IBM Systems Journal*, Vol. 21, No. 2, 1982, pp. 131-161, (IBM Reprint Order No. G321-5164)

John Seymour, *Coping With Computer Egos*, AMA Management Briefings, 1984, 52 pp., 135 West 50th Street, New York, N.Y. 10020

#### Other readings of interest

from IBM,           IC Documentation Examples, GG22-9268  
                      IC Administration, G320-6614  
                      External Data Access in an IC Environment, G320-6358  
                      IC Support Center Implementation Guide, ABOF-1857  
                      Systems and Products Guide, G320-6300

from AMA, "Effective Management of Information: How to Meet the Needs of All Users," *Management Review*, February 1984, pp. 8-14



# MANAGEMENT TOOLS FOR THE INFORMATION CENTRE

Prepared By  
GWEN R. WARLOW  
IBM CANADA LTD.

# AGENDA:

- 
- MISSION
- 
- ORGANIZATION
- 
- USER COMMUNICATIONS
- 
- ACCESS TO SERVICE
- 
- SYSTEM MANAGEMENT CONTROLS
- 
- SECURITY
- 
- SUPPORT
-

# INFORMATION CENTRE

---

## MISSION

“ TO MAKE IT EASIER FOR USERS TO USE COMPUTING RESOURCES TO IMPROVE THEIR SELF-SUFFICIENCY AND EFFECTIVENESS”.

- PROMOTION
  - GUIDANCE...NOT CODING
  - EDUCATION
  - PROBLEM ASSISTANCE
  - PLANNING
  - ENVIRONMENTAL SUPPORT
-

# INFORMATION CENTRE

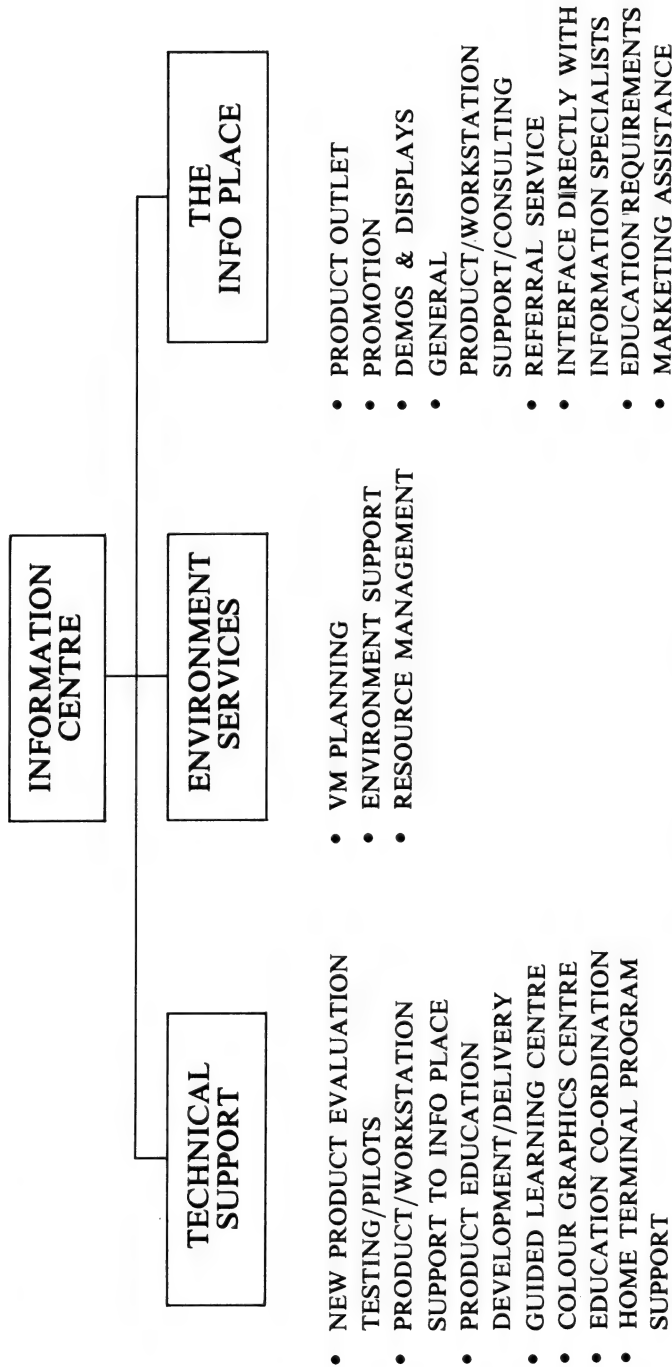
---

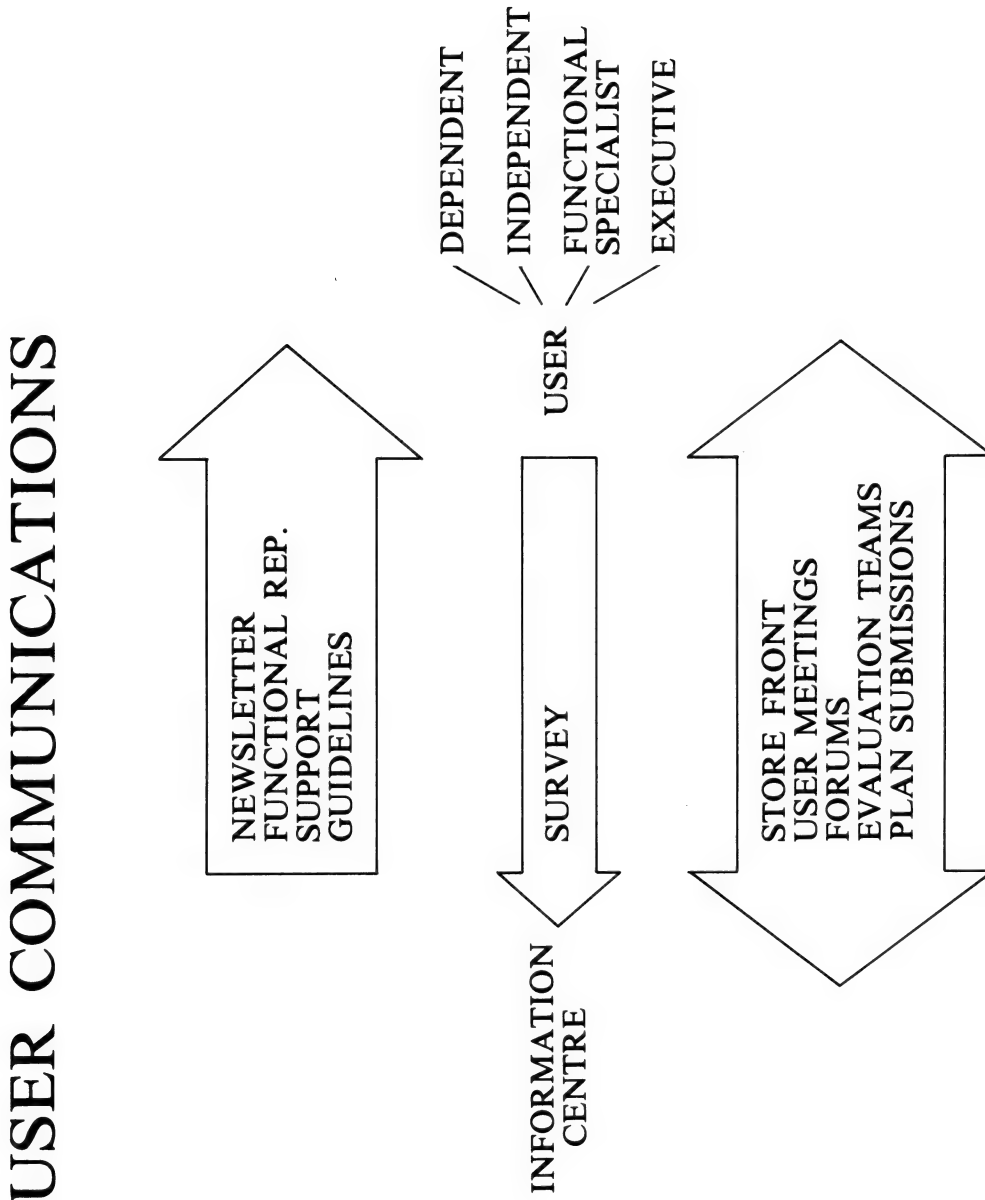
## GROWING RESPONSIBILITIES

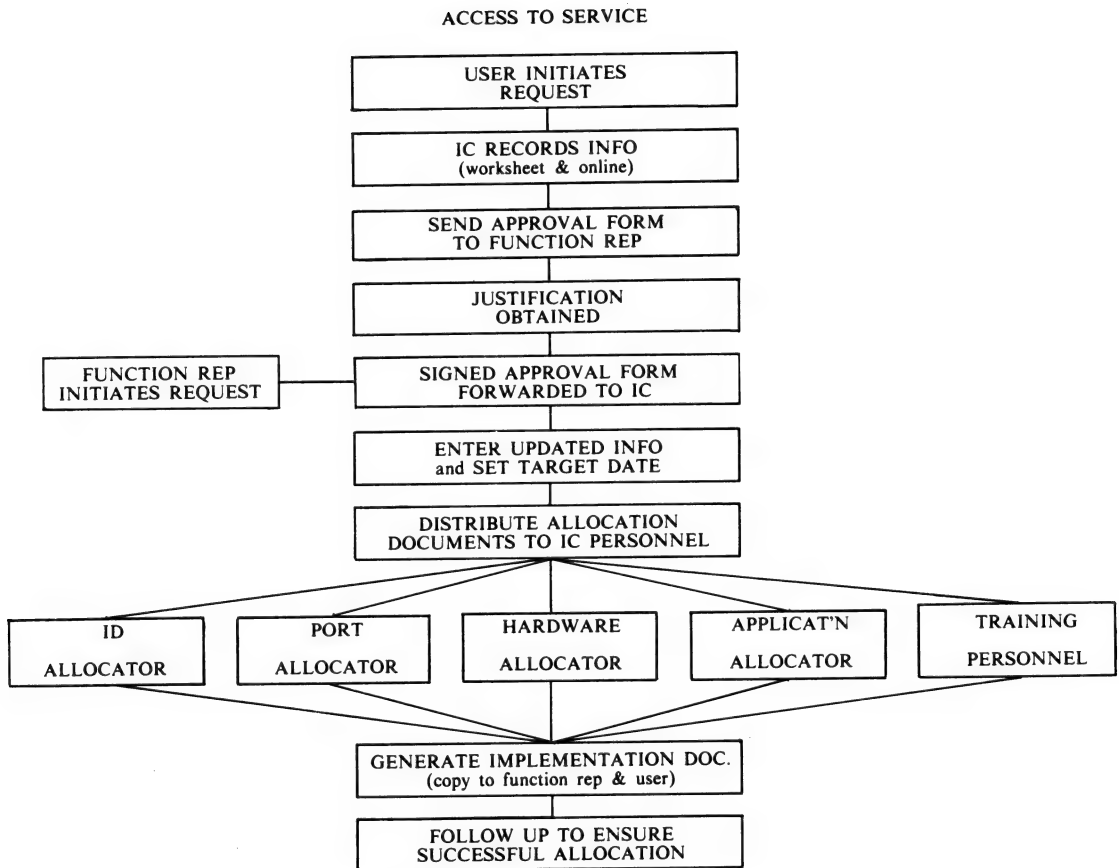
- RESOURCE MANAGEMENT
  - RESOURCE PLANNING
  - OFFICE SYSTEMS
  - “ON DEMAND” EDUCATION
  - OFF SITE SUPPORT
  - COLOUR GRAPHICS
  - WORKSTATION MANAGEMENT
  - HOME TERMINALS
  - “INFO PLACE” ENVIRONMENT
-



# INFORMATION CENTRE ORGANIZATION







# JUSTIFICATION

---

## BUILD A BUSINESS CASE

- BUSINESS CASE PROCESS
    - RECONFIRM REQUIREMENTS AND BENEFITS
    - EVALUATE ALTERNATIVE
    - RECOMMENDATIONS
  
  - BUSINESS CASE FORMAT
    - BENEFITS
      - HEADCOUNT AVOIDS AND SAVES
      - DISCONTINUED SYSTEMS
    - INVESTMENT
      - CAPITAL OUTLAY
    - OPERATING EXPENSES
    - NET AFTER TAX CASH FLOWS
      - PAYBACK
  
  - REVIEW/APPROVAL PROCESS
    - COMPARISON TO PLAN
    - CONTAINMENT
    - SIGN-OFF (FINANCE & FUNCTION)
-

# SYSTEM MANAGEMENT CONTROLS

---

- CHANGE MANAGEMENT
    - SINGLE FOCAL POINT
    - RIGID TEST PROCEDURES
    - WELL DOCUMENTED
  
  - PROCESSING MANAGEMENT
    - PRODUCTION WORKLOAD SCHEDULING
    - PRODUCTION DISTRIBUTION SCHEDULING
    - RESOURCE/DATA CONTROL
    - SERVICE LEVEL MONITORING
  
  - NETWORK MANAGEMENT
    - USER SUPPORT
    - COMMUNICATION
    - SINGLE FOCAL POINT
    - CONSOLIDATED APPROACH
    - STATUS
  
  - CAPACITY MANAGEMENT
    - PLANNING PROCESS
    - USER REQUIREMENTS INPUT
    - ACTUAL VS PLAN MEASUREMENT
-

# SYSTEM MANAGEMENT CONTROLS (cont.)

---

- PROBLEM MANAGEMENT
    - FORMAL PROCEDURE
    - AUDIT AND TRACKING
    - RESPONSIVENESS
  
  - RECOVERY MANAGEMENT
    - TESTED
    - WELL DOCUMENTED
    - ESTABLISHED PRIORITIES
    - USER AGREEMENT
    - PART OF THE PLANNING PROCESS
  
  - PERFORMANCE MANAGEMENT
    - EQUIPMENT UTILIZATION
    - TREND DATA
    - SERVICE LEVEL MONITORING
  
  - MANAGEMENT REPORTING
    - INFORMATION VS DETAIL
    - SERVICE LEVEL EMPHASIS
    - CONTROL/AUDIT
-

# PLANNING

---

## FORMAL PLANNING

- 2 YEAR OPERATING PLAN
- 5 YEAR STRATEGIC PLAN

## CLEAR LINES OF RESPONSIBILITY

- USER ORGANIZATIONS
  - DEVELOP REQUIREMENTS PLAN E.G. SERVICES, CPU HOURS, STORAGE, NETWORKS, TERMINALS
  - JUSTIFY COSTS
- COMPUTING CENTRES
  - CONSOLIDATE REQUIREMENTS
  - DEVELOP SOLUTIONS E.G. CENTRALIZED OR DDP
  - DEVELOP TECHNICAL AND CAPACITY PLANS
  - ALLOCATE COSTS

## SERVICE LEVEL CONTRACT

- CAPACITY AND RESPONSE TIME PLANS
-

# HOW DOES THE IC PARTICIPATE

---

- CHANGE CONTROL
  - MIGRATIONS
  - PILOTS
-



# HOW DOES THE IC PARTICIPATE

---

- CHANGE CONTROL
    - WEEKLY SCHEDULE OF ALL SYSTEM CHANGES
    - FILTER WHAT CHANGES DIRECTLY AFFECT USERS
    - NOTIFY IC PRODUCT SUPPORT
      - DETERMINE USER IMPACT
      - CREATE USER DOCUMENT
    - NOTIFY USERS
      - LOG MESSAGE
      - FUNCTIONAL REPRESENTATIVES
      - DOCUMENT AVAILABLE ON-LINE
      - NEWSLETTERS
    - PRODUCTION
-

# HOW DOES THE IC PARTICIPATE

---

- MIGRATIONS

- A) LOAD BALANCING

- MULTI PROCESSORS AT SITE

- CRITERIA:

- LOGICAL USER GROUPS
    - PREREQUISITE DATA AVAILABLE
    - USER LINKS
      - EACH USERID REVIEWED
    - USER TYPE
      - TRIVIAL/NON-TRIVIAL

- DECISION:

- FUNCTIONAL REPRESENTATIVES

- B) NEW SITE PROCESSOR

- CRITERIA:

- LOGICAL GROUP
    - SITE OCCUPANCY

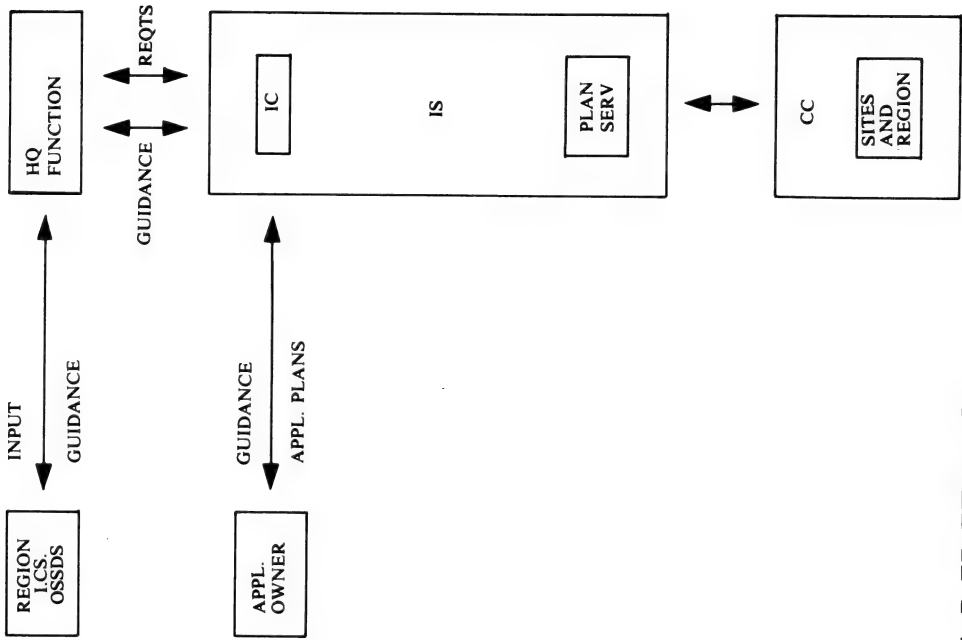
- DECISION:

- FUNCTIONAL REPRESENTATIVES
-

# HOW DOES THE IC PARTICIPATE

---

- PILOT
    - NEW PRODUCT/SERVICE ANNOUNCED
    - DETERMINE PILOT GROUP
    - DEFINE OBJECTIVES OF PILOT
      - USEABILITY
      - PERFORMANCE
      - USER FRIENDLINESS
      - QUALITY
      - POTENTIAL APPLICATIONS
      - BENEFITS
    - PILOT RESULTS
    - IMPLEMENTATION
    - POST IMPLEMENTATION REVIEW
-

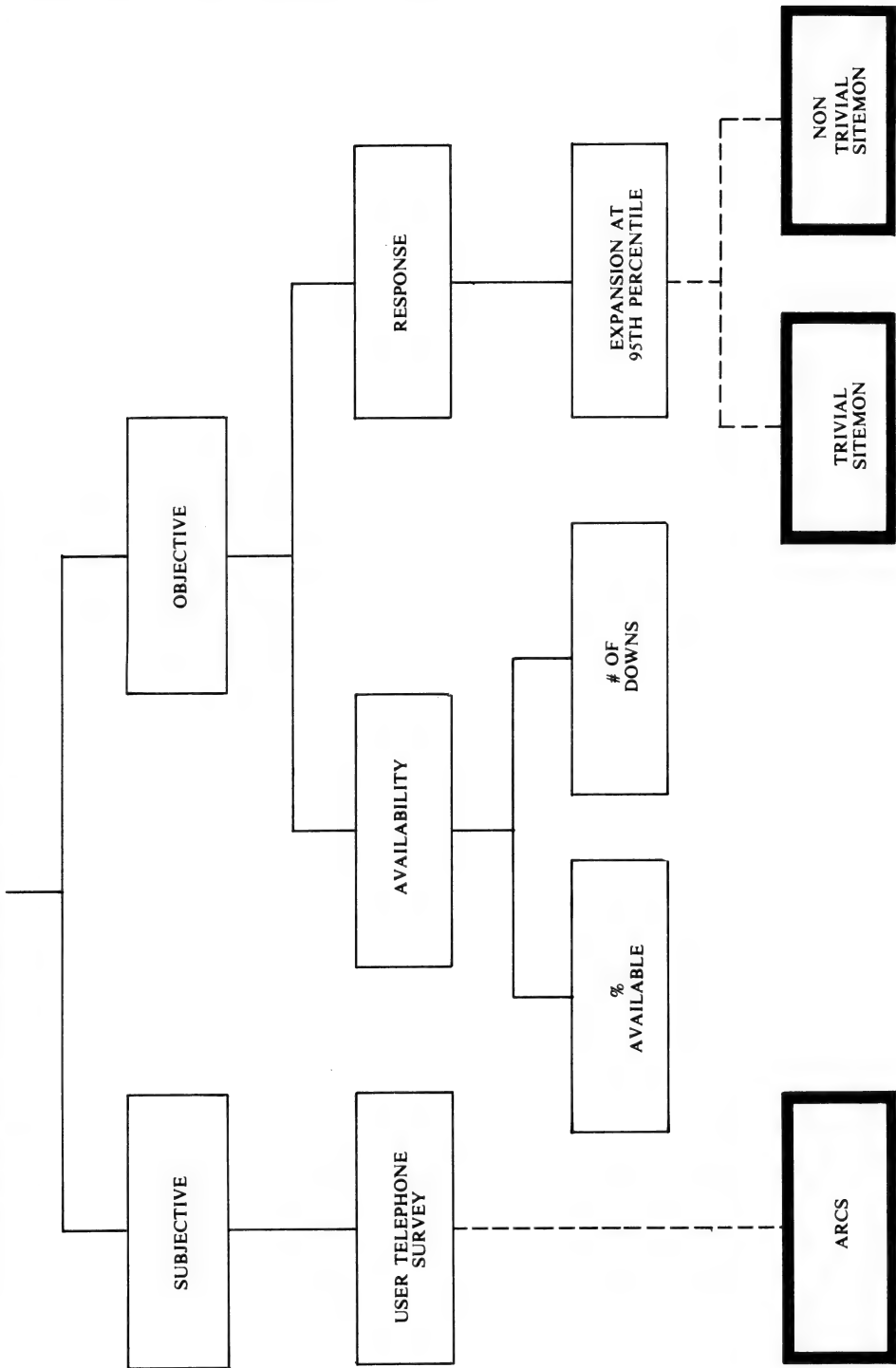


COMPUTING REQUIREMENT BY SITE/SERVICE

- CPU, DASD, TERMINALS
- BENEFITS
- AFFORDABILITY
- SERVICE DELIVERY STRATEGY
- GUIDANCE - CPU, DASD
- HISTORICAL USAGE DATA -MAS
- REASONABILITY CHECK OF INPUT
- CONSOLIDATION BY SITE/SERVICE
- TRACKING/REPORTING
- TOTAL CC PLAN INTEGRATION
- INTERFACE TO CC
- (SUBMISSION/NEGOTIATION)
- PRICING/RECOVERY/CHARGEOUT
- PLANNING SYSTEM (OPAR)
- NEW HARDWARE/SOFTWARE
- REQUIREMENTS
- BENEFIT CONSOLIDATION/
- VERIFICATION
- CAPACITY PLANNING
- TECHNICAL SOLUTION
- (HARDWARE SYSTEM SOFTWARE)
- IMPLEMENTATION/INSTALLATION

# END USER SERVICE PLANNING

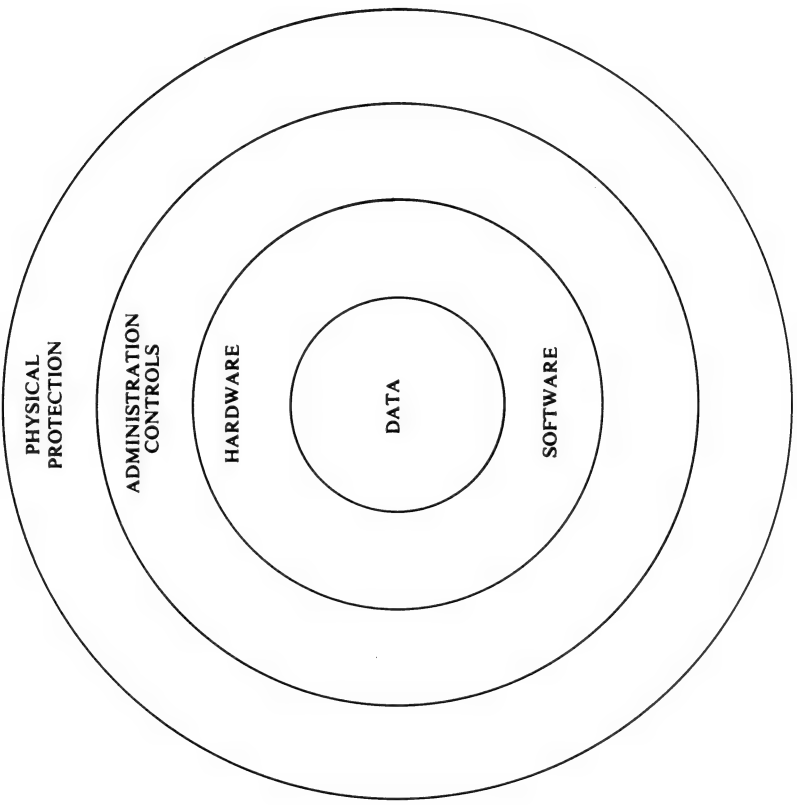
# DAILY SERVICE RATING



# CAPACITY PLANNING TOOLS

---

- RESOURCE MANAGEMENT SYSTEM
    - PEAKS
    - PLAN VS ACTUAL
  - TRACE
    - BILLING
  - MEASUREMENT ACCOUNTING SYSTEM
    - PRODUCT
  - SERVICE LEVEL REPORTED
    - DOWNS
    - RESPONSE TIME
  - DAILY SERVICE RATINGS
    - USER
    - SYSTEM
-



# SECURITY

IS : PROTECTION  
OF : DATA AND INFORMATION  
FROM : UNAUTHORIZED OR ACCIDENTAL DISCLOSURE, MODIFICATION, DESTRUCTION  
AND : RECOVERY  
OF : DP ACTIVITIES  
IN : CASE OF DISASTER

# SECURITY

---

## INFORMATION CENTRE

- DIVISION OF RESPONSIBILITY
    - DIRECTORY MAINTENANCE
    - LINK ACCESS
  - AUDIT TRAILS
    - HARDCOPY OF WRITTEN REQUESTS
  - CERTIFICATION TO USER
  - SEMI-ANNUAL VALIDATION
-



# SECURITY

---

## USER

### TERMINAL

#### ACCESS

- WRITTEN REQUEST
- REGULAR PSWD CHANGED
  - UNIQUE
- ONLY MANAGEMENT APPROVED PURPOSES

#### APPLICATION/SOFTWARE

- ACCOUNTABLE FOR ALL USE/MISUSE

#### DATA

- CLASSIFIED
- ENCRYPTION
- MACHINE ATTENDED
- ALL TAPES KEPT UNDER LIBRARY CONTROL

### PERSONAL COMPUTER

- ONLY MANAGEMENT APPROVED PURPOSES

- COMPLY WITH LICENSE AGREEMENT
- COPY ONLY IN COMPLIANCE WITH AGREEMENT

- CLASSIFIED
  - ENCRYPTION
  - MACHINE ATTENDED
  - DISKETTES KEPT UNDER LOCK AND KEY
  - REMOVE DISKETTES FROM SYSTEM WHEN NOT IN USE
  - CAREFUL HANDLING
  - PROPER DISPOSAL OF DISKETTES
-

# SECURITY

---

## USER (cont.)

### TERMINAL

#### PHYSICAL

- INSTALL WHERE CASUAL AND UNAUTHORIZED ACCESS IS PREVENTED BY BUILDING SECURITY
- LOGGED OFF WHEN UNATTENDED

#### BACKUP

- DAILY BACKUP OF USER DATA
- ADDITIONAL BACKUP AND OFFSITE STORAGE OF SPECIFIC VITAL RECORDS

#### RECOVERY

- RESTORE OF DATA

### PERSONAL COMPUTER

- INSTALL WHERE CASUAL AND UNAUTHORIZED ACCESS IS PREVENTED BY BUILDING SECURITY
- TURNED OFF WHEN UNATTENDED
- XT LOCKED IF CLASSIFIED INFORMATION STORED

- BACKUP DISKETTES REGULARLY

- FROM BACKUP DISKETTES
-

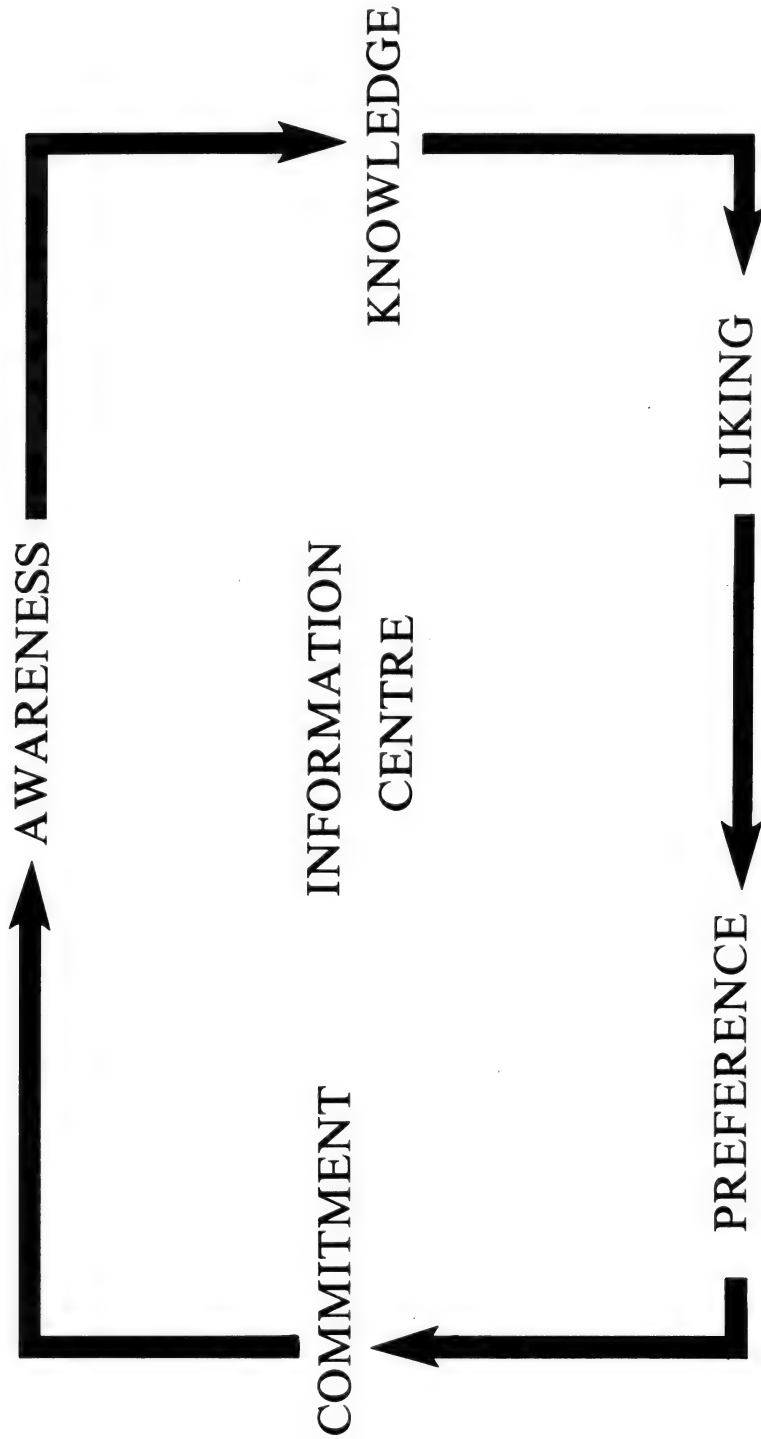
# SECURITY

---

## COMMUNICATIONS VEHICLE

- DOCUMENTS
    - GUIDELINES
    - STANDARDS
  - EDUCATION
    - SEMINARS
  - ON-LINE REMINDERS
    - LOGO
    - CERTIFICATION
  - STICKERS
  - BROCHURES
-

## SUPPORT CYCLE



	AWARENESS	KNOWLEDGE	LIKING	PREFERENCE	COMMITMENT
FUNCTIONAL SPECIALIST	DEMO	SEMINAR			
	PROMOTION	REFERENCE LIBRARY			
	WHAT'S NEW	APPLICATION CATALOGUE			
				WORKSHOP	
					SWAP SHOP
		CONSULTING			
				OUTLET	
				SUPPORT	
				FOLLOW UP	
EXECUTIVE	EXECUTIVE DEMO	EXECUTIVE BRIEFING		EXECUTIVE PACKAGE	
	PROMOTION			OUTLET	
	WHAT'S NEW			SUPPORT	
				FOLLOW UP	



# **LOGOS: An APL Programming Environment**

David B. Allen

Mark R. Dempsey

Leslie H. Goldsmith

Kevin L. Harrell

I.P. Sharp Associates

## **Abstract**

Programming environments support the creation, maintenance, execution, and deployment of programs. The development of complex application systems written in APL has always been hampered by the lack of a comprehensive programming environment for the language. In this paper, a new approach to software creation and management is presented through the use of LOGOS, an integrated software development environment for APL. LOGOS provides support to the user, developer, and manager of application systems. Its centralized and uniform structure removes much of the drudgery of dealing with software, encourages sharing of ideas and programs among users, and leads to increased programmer productivity and improved software reliability.





APL has long been recognized as a powerful, dynamic, and high-productivity language in which system development and testing can be performed with great speed. However, compared with other systems, even as the *language* excels, its programming *environment* is archaic and limited. Large systems are difficult to prepare and audit during their development, and the integration and installation of software can be tantamount to a nightmare.

With LOGOS, things needn't be that way. LOGOS is a comprehensive programming environment in which APL application programs or systems may be conceived, developed, maintained, and distributed. It is not just a collection of useful tools, but it is an integrated software development environment in the true sense of supporting and aiding management of software throughout its life [Howd82, Deli84]. The term 'integrated' emphasizes the difference between a loosely coupled system and an environment in which the components are designed cohabitatively. LOGOS presents a consistent user interface paradigm which, in addition to being simple and extensible, buries many of the mental context switches normally associated with moving from one aspect of software maintenance to another. In addition, facilities and tools in LOGOS are designed to amplify the effects of other facilities, thereby yielding a synergistic effect over the system as a whole.

Programming environments exist for many systems other than APL. Examples of well-known environments include Interlisp [Teit81], UNIX [Kern81, Ritc78], Smalltalk [Goldb84], and Gandalf [Medi81]. Several non-integrated APL packages which specialize in certain aspects of software manipulation have been written. For example, I.P. Sharp's APE [IPSA83] provides good control over the creation and revision of modules, and The Planning Economics Group's Workshop [PEG81] handles dynamic materialization of objects into a workspace. But there is a paucity of viable products which provide a full spectrum of support to the programming task as a whole. In fact, we know of no such application written for APL other than LOGOS.

In general terms, software can be traced through a series of phases which are collectively known as the *life cycle* [Jens79]. The basic elements of the life cycle are specification, design, coding, testing and debugging, integration and distribution, and maintenance—the latter being repeated cycles of the previous steps. LOGOS supports activities during all phases of the life cycle, with particular emphasis on those involving complex interactions between the user and the programming environment. The efficacy of LOGOS in dealing with these disparate phases makes it not only a means for software development, but also an end.

LOGOS normally runs in your own application workspace, so that it can directly participate in operations affecting the environment global to it. The system contains a centralized data base which provides a hierarchical structure in which arbitrary APL objects may reside. Protection controls in the data base ensure no unauthorized accessing or contamination of information. The protection controls are fine enough to grant a particular user the right to execute a specific function, without giving any access whatsoever to other objects.

Very simple commands let you move objects into LOGOS. Once there, you can set access controls to limit or extend the degree to which your objects may be shared and disseminated. The system immediately begins keeping track of who (including you) is using your data, so that you can assess the implications of modifying or removing an object *a priori*. If desired, backup copies of each object you change can be kept automatically and recalled at any time.

Sophisticated tools let you build systems in LOGOS and statically analyse what you've got at any time. These tools can operate on objects within LOGOS, or on objects within the workspace for simplified debugging. There is an integrated interface to WSDOC [IPSA84] to permit rapid and comprehensive documentation of programs and variables.

When your application is ready to run, LOGOS provides the facilities necessary to produce production workspaces and files from your source programs and data. Where the integrity of a running application is involved, you can easily back up existing workspaces and files prior to generating new ones. If your application involves more code or data than comfortably fits in a workspace, LOGOS can assist you by creating a 'page file' from which objects may be fetched when required. The page file design can be completely under LOGOS's control if you choose; alternatively, you can provide input to the file generation to allow LOGOS to take advantage of the specific structure and usage patterns of your application.

## 2. SCHEMATIC VIEW OF LOGOS

The structure of a development environment is a critical factor influencing how productive and useful the environment is perceived to be. In the design of LOGOS, three guiding organizational characteristics were adopted:

1. There is a central shared data base, or file system, which stores all of the information in or relating to LOGOS. This includes arbitrary user programs and data, tools to assist in the execution of certain programming tasks, and the functions which comprise LOGOS itself.
2. The domain of the conventional APL workspace is, by itself, too restrictive to permit the operation of general, medium- or large-scale applications. The concept of a work area needed to be extended to include a companion arrangement of objects available to the workspace but not necessarily co-resident (namely page files), and a permanent, high-volume storage medium (namely data files). This conceptual refinement had to be carefully integrated with the essential aspects of familiar workspace interaction, so that the extension was both natural and easily exploited.
3. A rich set of carefully designed, user-friendly commands is required to support the bi-directional flow of information between the central data base and the extended workspace. These commands had to be able to handle the management of information

throughout the iterative evolutionary and development processes attendant with software generation. Moreover, the commands had to be able to work together, syntactically and semantically, so that the result of one could function as the argument of another.

Figure 1 shows a simplified structural overview of LOGOS. From the end application's perspective, the appearance is rather like that of any conventional APL system: one or more workspaces cooperating (often invisibly) with data files and page files. Simpler applications may include only data files, only page files, or no file interaction at all.

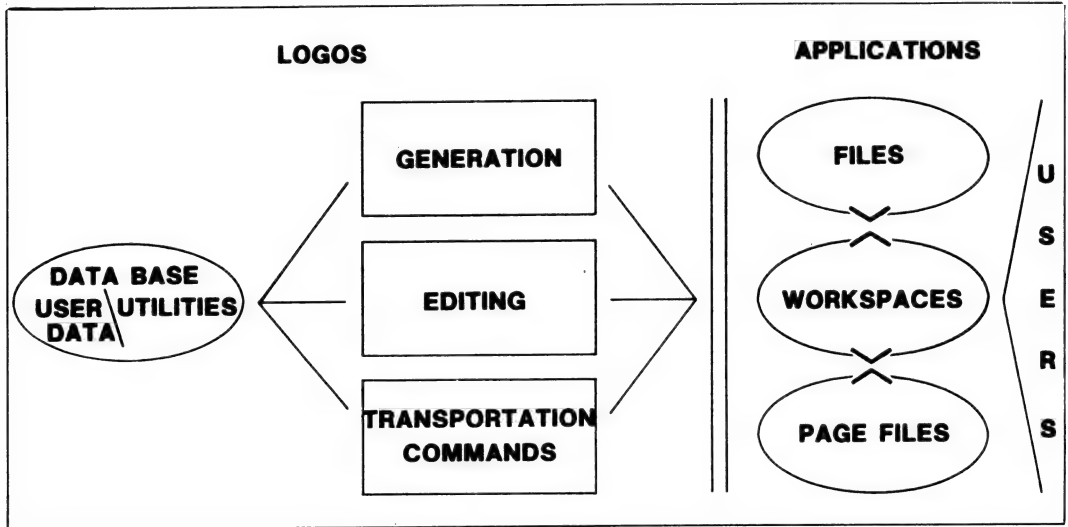


Figure 1

LOGOS supports the end environments that make up the application by providing tightly integrated, powerful mechanisms for generating and maintaining workspaces and files. This is made possible through software which is intimately familiar with APL data types and structures. The LOGOS editor, for example, can operate on any object, regardless of data type or rank.

Objects can move between the LOGOS file system and applications by generating or regenerating a file or workspace; by editing an object either in LOGOS or in a file or workspace; or by explicitly moving objects using transportation commands designed for this purpose. In the case where movement occurs under the aegis of the editor, LOGOS's knowledge of

where objects are used is particularly helpful, for it makes it possible to alter an object within LOGOS and have that change take either immediate or deferred effect upon the applications which reference it.

### 3. THE OBJECT DATA BASE

The object data base is one of the most significant aspects of the design of LOGOS. Just about any interaction with LOGOS makes some use of the data base, and processes such as generation use it extensively. As Osterweil points out [Oste81], "if software projects can be thought of as coordinated efforts to gain and disseminate a highly structured body of knowledge about a problem and its solution, the progress of the project will be best assured and facilitated by capturing, structuring, and disseminating that body of knowledge as faithfully and as effectively as possible." It is therefore not surprising that at the centre of a programming environment is a comprehensive data base of information about the software it manages.

In LOGOS, the data base consists of a unified, flexible file system which may contain arbitrary APL objects. The file system is a single-rooted, hierarchical tree whose interior nodes define directories and whose leaves define either directories or objects (see Figure 2). The single-rooted aspect of the file system is significant, for it creates an environment wherein sharing and communication are encouraged through provision to get from any node to any other (access rights permitting). Moreover, the system's hierarchical structure closely parallels the homologous hierarchy of concepts which reflects the way we think and organize information [Golds84].

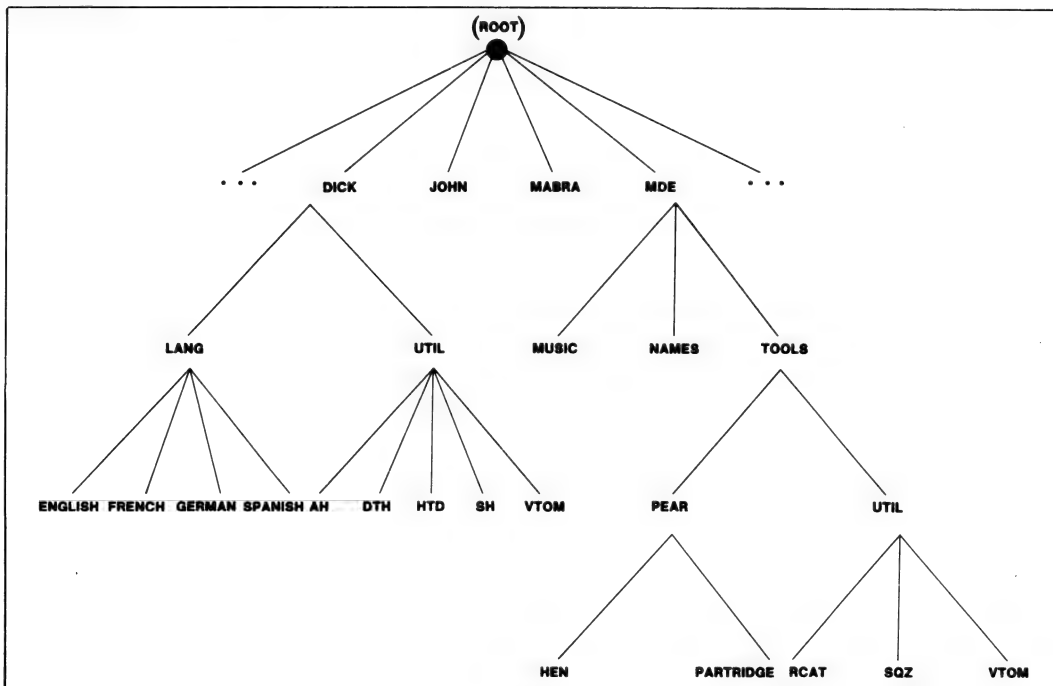


Figure 2

At the level below the root of the file system, there is an entry for each user in LOGOS. A user is named via an alphanumeric *alias*, and more than one alias may be associated with the same user. Although, by convention, one alias usually represents a form of the person's name, other surrogates may refer to projects or categories with which the person is associated. This provides the capability to divorce a person's identity from a product's.

Below each alias is an arbitrary structure of directories and objects, established by the user. An object is, simply, any single thing which may exist in a workspace or file (except a group). For example, a program, a numeric matrix, a package, and a nested array are all examples of objects. There are no restrictions placed on the composition of objects within LOGOS. Application programs may expect an object to have a particular structure; however, this structure is imposed by the programs which use LOGOS, and not by the system itself.

### 3.1 Paths and Names

Any directory or object can be accessed by its name. When the name is specified to, or echoed by the system, it may be in the form of a *pathname*, which is a sequence of directory names separated by dots (.) and optionally ending in an object name. If the sequence begins with a dot, the name is taken to be relative to the root of the file system. For example, in Figure 2, the name *.DICK.UTIL.VTOM* is interpreted by beginning at the root and traversing the directory *DICK* for the directory *UTIL*. *UTIL* is then, in turn, searched for the name *VTOM*. *VTOM* may be an object, or it may itself be a directory containing other objects. Notice that *DICK*'s *UTIL* directory is unrelated to the one found along the path *.MDE.TOOLS*.

A pathname not starting with a dot is assumed to be relative to the user's current *working directory*. Thus, the name *UTIL.VTOM* specifies *VTOM* within *UTIL* within the current working directory. The current working directory is controlled by the command *WORKDIR*, and may be modified or inquired at any time. For example:

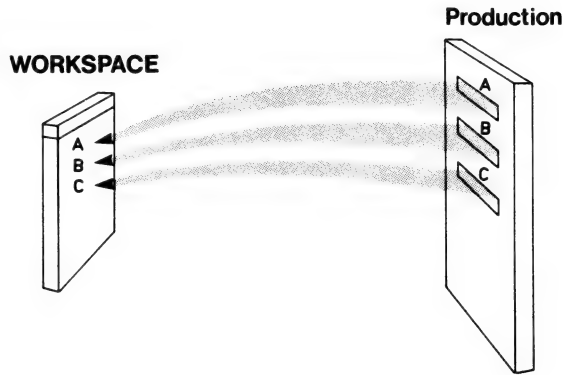
```
◇ WORKDIR
.DICK
```

(The "◇" character is the LOGOS prompt, and is also the character you can use to separate multiple commands on the same input line.) When the working directory is changed, *WORKDIR* returns its new value:

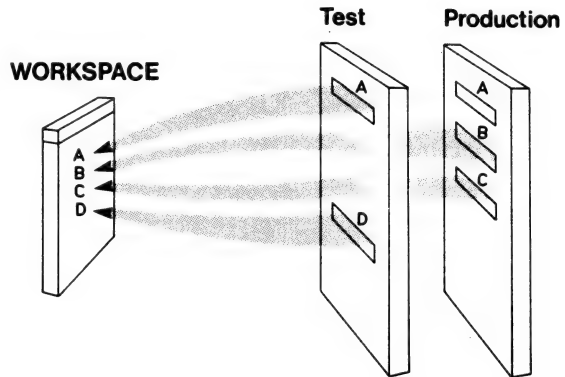
```
◇ WORKDIR .UTIL
.DICK.UTIL
```

You can establish multiple working directories—a facility which has enormous power in offering directory threading or overlaying capabilities, as illustrated in Figure 3. When you select more than one working directory, the directories are searched in the order you specify them, and the first instance located is the one which is returned.

A 'production' application being generated from LOGOS. In this instance, all of the objects in the workspace are obtained from a single working directory. The objects represent the released versions of functions or variables which comprise the production application.



A test version of the same workspace. Both the TEST and PRODUCTION working directories have been specified, in that order, causing LOGOS to give precedence to those objects found in TEST. The generated workspace will contain any modified or new objects from the TEST directory, along with any remaining objects from PRODUCTION. When the test version has been verified, the COPY command may be used to incorporate the changes and additions into the production directory.



A more complex structure. Any combination of working directories may be specified to achieve the desired overlaying of objects. In this example, a private set of modifications is added to the front of the working directory specification. Again, the precedence ordering of the directories favours the first directory in which an object is found. This structure is particularly efficacious when more than one person is working on the development or maintenance of a project.

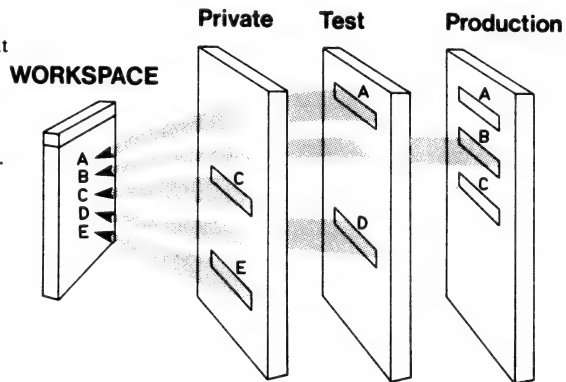


Figure 3

Object type and version qualifiers may be specified using an extended pathname format, thereby permitting these parameters to participate in the selection criteria for primitive searches. For example, *UTIL.HTD[F]* selects the function *HTD* within the directory *UTIL*, *UTIL.HTD[8]* selects version 8, and *UTIL.HTD[F8]* combines both criteria.

The same object or directory can appear in several directories under different pathnames. This feature is called *linking*, and it permits you to create a link, or reference, to a path elsewhere in the data base [DGC82]. This is particularly convenient if you regularly have cause to use a program of someone else's, and would like to be able to refer to it as if it were your own. (You could copy the object, but this would create two disjoint versions of the source, thereby complicating future maintenance efforts.)

### 3.2 Pathname Shorthand

The file system provides a convenient shorthand for generating a sequence of names which match a particular pattern. For example, *LIST TEMP?\** will print those names in the current working directory which begin with *TEMP*. "?" stands for "match any character," and "\*" means do so as many times as possible. The pathname *.DICK.UTIL.??* finds all two-letter names in *.DICK.UTIL*:

```
◇ LIST .DICK.UTIL.??
AH
SH
```

Other, more complex patterns are possible (see [Demp81] for details). The mechanism is useful both to save typing and to select families of names. It encourages systematic naming of directories and objects, and it makes it possible to process sets of names as easily as single ones.

Patterns and multiple working directories work well in conjunction with each other. If the different working directories contain some of the same object names, then a command such as *LIST ?\** will find the unique names, giving preference to the first match encountered where a name appears in more than one directory.

### 3.3 Object Attributes

An aspect of programming environments which both theory and practice have demonstrated to be of significant value is the notion of a central information repository, in which all of the data essential to an object is collected and stored [Wass81]. In LOGOS, the APL definition of an object is only one of a myriad of attributes which describes it. Other information which the user may specify includes documentation, a change journal to record

what modifications were made and why, access controls to limit availability (Section 3.4), compilation directives to massage the source into a transformed entity for production use (Section 5), and a broadcast note to alert potential users of the object (Section 9.3).

### 3.4 Access Controls

Security is the protection of data against accidental or malicious disclosure, and integrity is the protection against accidental or malicious modification [Gold81]. The internal design of LOGOS carefully guards against unauthorized access to or contamination of data, and also provides self-protection logic to ensure that data is available and consistent after events such as a system crash or line drop.

The access controls in LOGOS are extremely powerful, permitting you to levy constraints at *any* node of the hierarchy. The constraints take the form of an *access matrix*, which stipulates users who may access the node, plus the types of permission to which each is entitled. Permission to read, write, execute, and control access may be assigned independently or in combination. When applied to a directory, the access settings take on a slightly different meaning; write, for example, means the ability to create or delete objects in the directory. A directory or object which is not available to a user is impervious to any searches performed by that user.

Two simple commands, *GRANT* and *REVOKE*, are used to modify access. Issuing the command:

```
◇ GRANT LHG UTIL.AH +PERM=RX
```

gives *LHG* read and execute permission to the path *UTIL.AH*. The "+" indicates that what follows is a command modifier, the name of which is *PERM* here. By selecting it, we overrode the default permission that *GRANT* extends, which is execute access only.

Access can be extended to individual users, as well as to *groups*, which specify the identity of users having a common affinity. Since groups are maintained on a system-wide basis, altering the definition of a group automatically affects the access rights of each node which subscribes to the group. This makes it easy to administer personnel changes on projects where sensitive information is involved.

## 4. SAVING OBJECTS IN THE LOGOS DATA BASE

LOGOS provides a variety of methods for putting objects into the object data base. They include a command to save objects, an option for the editor, and a command which can analyse and selectively transfer objects from your workspace environment to the file system.



The fundamental method for saving objects is the *SAVE* command. The command line:

◇ *SAVE REPORTGEN .DICK.UTIL.HTD*

saves the definitions (the *source*) of *REPORTGEN* and *HTD* from your active workspace environment into the file system. *REPORTGEN* is saved under your current working directory, and *HTD* is saved in the explicitly specified path. Optional modifiers allow you to set any of the other object attributes, such as the broadcast note or the compilation directives, separately or in combination. The capabilities of this command are also available through stand-alone utility functions, which can be called from APL programs outside of the interactive LOGOS interface.

The LOGOS object editor provides another means of entering an object into the file system.

The editor has integral access to the LOGOS data base and your workspace environment, and can edit or create any object.

Finally, you can use the *SNAP* command to put objects into LOGOS. *SNAP* provides capabilities which are fundamentally the inverse of workspace generation (see Section 6). It puts objects from the execution environment into the file system, and optionally builds a script capable of reproducing the workspace. If some objects being snapped are already resident in LOGOS, then only those whose definitions have changed are actually resaved. *SNAP* makes it simple to add new workspaces to LOGOS, and to update the data base with changes made outside of LOGOS.

Once an object exists in the LOGOS data base, it is subject to all of the capabilities offered by the LOGOS environment. It can be moved, copied, edited and analysed. Access controls can be set to restrict who may use the object, and how the object may be used. And, of course, the object may be employed in the construction of end application environments.

## 5. END ENVIRONMENTS—WHAT LOGOS BUILDS

A most important facet of LOGOS is its ability to produce end environments, which are manifestations of application systems consisting of programs and data. An end environment is generally built around a workspace and some data files. The workspace usually contains the program code and global variables necessary to implement the system. If necessitated by space limitations or name usage problems, some of these workspace objects might be stored in a file, and retrieved as needed. As well, permanent or shared data are often stored in APL files.

The production of end environments is known as *generation*, and workspaces and data files are specified through *generation scripts*. A script is a program which may contain both APL statements and LOGOS commands. Because it is essentially an APL function, it can easily express any transformations required between stored LOGOS objects and a fully functional end environment.

While workspaces and data files alone are often sufficient to implement an application system, many large systems can run into problems due to workspace size limitations. LOGOS provides a practical solution to this problem through facilities which can dynamically define functions and variables from files for execution, and expunge them when they are no longer needed. This technique is known as *paging*.

Paging allows you to simulate a virtual working environment. A system which uses paging can benefit from many extensions to traditional workspace technology. Potential benefits include:

- Workspace capacity limitations are largely eliminated. Only those objects which are required at a particular instant need be present in the workspace.
- Integration of disparate subsystems is simplified. Name conflicts between subsystems can often be resolved so that the appropriate objects are associated with the intended subsystem.
- Isolation and security of system components is possible through the use of *shells*. A shell is a function which establishes a protected environment around its constituent objects. Shells are both transportable (they can be copied into a workspace as a single object), and self-contained (all but the shell function itself evaporates when its execution terminates).

LOGOS directly supports a full gamut of paging activities. An object may be paged in explicitly by program request, or implicitly by its absence in the workspace. It may be fetched by itself when required, or *en masse* with its calling tree, which can be computed automatically and to an arbitrary depth. The controls available for the disposal of objects no longer needed are also versatile. In short, almost any system can run comfortably within the LOGOS paging environment.

It is often desirable that the source version of an object undergo some form of transformation before being introduced into an end environment. For example, if the object is a function, you may want its comments removed or its statements 'diamondized'; or you may wish to have evaluated data (such as file names and pass numbers) inserted at generation time. If the object is a variable, you may want to have it processed by a text formatter if it represents a textual description. To support these requirements, LOGOS provides extensive object *compilation directives*. A full range of compilation directives, some which are user-specifiable, is provided. With compilation, you can keep your stored objects in a format most convenient for you, and let LOGOS create objects which are most effective in your end environment.

## 6. GENERATING ENVIRONMENTS

As has been mentioned, application systems are created from the object data base through the generation process. An application system may consist of any combination of workspaces, data files, and page files. It can be generated using a combination of LOGOS commands, generation scripts, and page documents.

### 6.1 Scripts, Workspaces, and Data Files

A generation script is an APL function with embedded LOGOS commands. Each command begins with a right parenthesis prefix to distinguish it from executable code. Here is a simple generation script to build a workspace:

```

▽ SAMPLEWSGEN
[1]  ) GENERATE A WORKSPACE FROM DIRECTORY 'SAMPLE'
[2]  →(18>1+3+□TS)↓OK ) DON'T GENERATE IF BEFORE 6 PM
[3]  'NOT BEFORE 6 PM' ◇ →0 ) PRINT REJECTION MESSAGE AND EXIT
[4]  OK: 'GENERATING <SAMPLE> WORKSPACE' ) PRINT A MESSAGE
[5]  )GET .LOGOS.SAMPLE.WORKSPACE.?* ) GET ALL OBJECTS
[6]  )WSSAVE SAMPLE ) SAVE WORKSPACE 'SAMPLE'
▽

```

This script will generate a workspace named *SAMPLE* containing the objects in directory *.LOGOS.SAMPLE.WORKSPACE*, unless the obscure restriction of 6 PM is not met.

Scripts are invoked by the *PROCESS* command. They can be niladic, or they can take arguments which are passed from the invoking command line. The ability to convey information from the user to the script aids in the generation of test or alternate versions of a system. By passing arguments, you can establish dynamic parameters such as workspace names, file names, or working directories, without any modification to a suitably structured script.

A command such as *GET*, used above, is an ordinary command which need not be embodied in a script. If it is, however, a permanent record of the functions materialized by it on behalf of the script is maintained. This is useful both for auditing, and for permitting fast updating of workspaces with only the changed or new objects.

The APL task with which commands interact is called the *execution environment*. By default, this is the active workspace. You may, however, use the *ATTACH* command to establish a secondary task. This task then becomes the repository for *GET* and the source for *SAVE* and the editor. For communication between the active workspace and the attached

execution environment, the *XFER* command may be used to transfer objects, and *EXECUTE* permits a dialogue. These facilities provide isolation between processes, which can be crucial in insuring consistent and meaningful results.

Because scripts are APL functions, they can also generate data files. Certain aspects of file generation do seem to require more programmer effort than necessary; for this reason, LOGOS provides a set of utility functions with which the programmer can save considerable time and effort. These utilities provide a convenient interface to the APL File System. They simplify many types of operations by automatically detecting and correcting common error conditions (such as *FILE FULL*), and by facilitating aggregate operations (such as file copy and the 'tie-or-create' style of opening a file).

## 6.2 Paging

Page files are generated by the *PFGEN* command. A simple example of page file generation might be:

```
◇ PFGEN +FILE=SAMPLE +PAGE=.LOGOS.SAMPLE.WORKSPACE.?*
```

This generates a file named *SAMPLE*, which contains every object in the *WORKSPACE* subdirectory. To use this file in an application, several LOGOS paging utility functions are helpful. A utility called *ΔLPAGEFILE* establishes the files from which objects are paged. Another, *ΔLPAGEIN*, materializes new pages, expunging old ones if additional space is required. Other utilities manage page-out, provide paging analyses, and assist in debugging systems which run with paging.

To establish an environment in which the above example will function equivalently to the workspace generated by *SAMPLEWGEN*, the following commands can be issued:

```
◇ ATTACH ◇ EXECUTE □LX←'ΔLPAGEFILE ''SAMPLE''
◇ GET ΔLPAGEFILE ΔLPAGEIN □TRAP +WORKDIR=.LOGOS.UTILITY.PAGING
◇ WSSAVE SAMPLE ◇ DETACH
```

(Of course, if you perform this sort of operation frequently, you could streamline this procedure by use of a simple script.) The global *□TRAP* defined by the *GET* command in the example assumes control each time an identifier is used without a corresponding referent object being found. The first time each object is referenced, it is defined by *ΔLPAGEIN* through the event trap so activated.

If a system consists of many, mostly self-contained functions, this approach is fairly efficacious. However, if the application has a deep function call tree, defining objects incrementally results in poor performance due to excessive paging activity. To counter this problem, an option to *PFGEN* tells it to compute function call trees to any depth and to cluster these with the root functions. This reduces paging activity considerably: when an object is paged in, so are the functions and variables it uses.

Paging is most useful when a system grows too large to be entirely workspace-resident, or when more than one system must simultaneously coexist in a workspace. Since the specification for such large or combined systems can be rather complex, LOGOS uses a *page document* to permit an arbitrarily sophisticated paging system to be defined. Page documents are invoked by *PFGEN*. Using them, pages can be named, so that they may be explicitly requested by your application, and pages can be defined to overlay other pages. A *base page* may be specified; this page is brought into the workspace upon page file initialization. Furthermore, files need not actually be dedicated to page files, but may contain a mixture of page areas and user data areas. This enables paging-based systems or subsystems to be developed independently and later integrated into a single application system.

Page files generated by *PFGEN* are fully integrated into the LOGOS environment. While working with your paged system, you can use the editor to modify objects, regardless of whether or not they are in the workspace. You can automatically edit the source objects, have the revised versions stored in the file system, and have the compiled versions automatically replaced into the paging areas in use.

A paged system can establish search lists of paging areas, as shown in Figure 4. Not only does this permit test versions of pages to be selected in preference to production versions, but it does so without any disruption to the production paging system. Testing and development are more efficient because they take place in the context of the full application system, without necessitating special test drivers or other debugging artifacts.

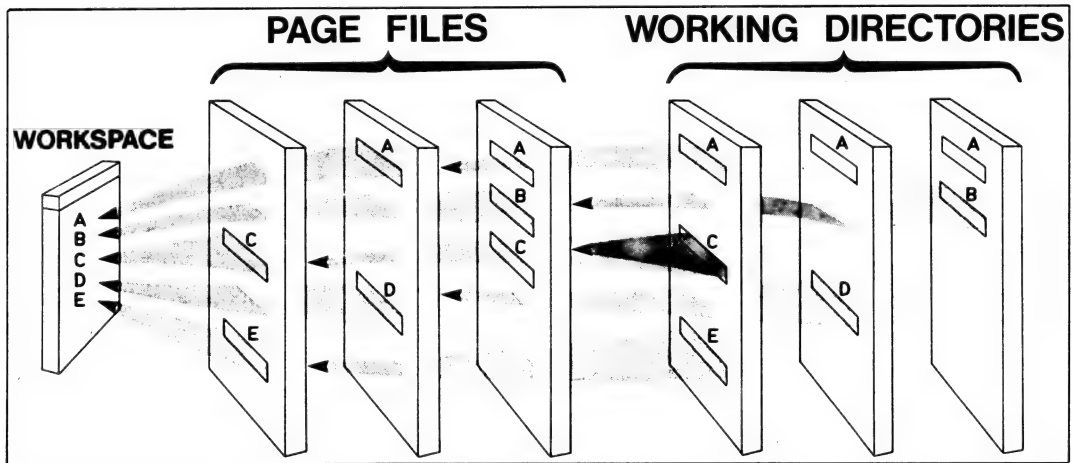


Figure 4

**Page files and working directories.** Multiple page files or paging areas, together with multiple working directories, permit an arbitrarily complex selection of objects to be materialized in your workspace. At one level, the working directories overlay to produce a selection of objects which can be used to build a page file (or, as shown in Figure 3, a workspace). Then, during dynamic execution of an application, multiple page files overlay to produce a selection of objects which are materialized in the workspace. The ordered precedence scheme of working directories applies to page files as well, so that the first occurrence of an object located in a series of paging areas is the one which is materialized. This layering provides an extra level of flexibility during the running of an application, and makes possible the dynamic choice of paging areas based upon criteria such as the user who is running the application.

## 7. NATIVE DEVELOPMENT TOOLS

The software tools provided by LOGOS are an important part of the integrated development environment. A powerful and extremely versatile editor, an integrated interface to WSDOC, generalized pattern search and replacement capabilities, and supplied utility functions form the cornerstones of a productive workbench for programmers.

### 7.1 The Object Editor

The LOGOS object editor is invoked by the *EDIT* command. It can edit any APL object, regardless of type or rank. It can even edit collections of objects, such as a workspace or the contents of a LOGOS directory. It has a full-screen user interface for certain devices, and a line- or window-oriented interface for any device which can be attached as a terminal to the SHARP APL system.

Objects to be edited may come from your workspace, from the LOGOS file system, or from paging files. References to compiled objects in end environments generated by LOGOS automatically link you to and allow editing of the source versions in LOGOS. Upon completion of editing, both the source and the compiled objects can be directly replaced into the environments from which they came.

The editor provides a powerful and consistent user interface which uses standard LOGOS command syntax. Its capabilities include:

- Pattern search and replacement
- Block move, copy, and delete
- Modification of object name, pathname, type, rank, and shape
- Positional or associative control over operations
- Simultaneous editing of multiple objects

The editor includes special support for arrays of enclosures and for packages. In addition, you can edit the ancillary attributes of a LOGOS object, such as its internal documentation and broadcast note.

### 7.2 Interface to WSDOC

The SHARP APL Workspace Documentation Facility [IPSA84], which produces a static analysis of a collection of APL objects, is directly accessible through the *WSDOC* command. Among the reports which WSDOC can produce are object summaries, display of functions and variables, cross-references, and function calling trees. This highly flexible repertoire of informative options makes WSDOC a most useful tool for the programmer's workbench.

You can directly apply the *WSDOC* command to any workspace or combination of LOGOS objects, or to any system generated by LOGOS. You can store your favourite choices of WSDOC options as objects; these stored configurations make the analysis process almost automatic.

### 7.3 Pattern Matching

Pattern matching (described in [Demp81]) is used in pathnames, in the object editor, and in the *LOCATE* and *REPLACE* commands. These commands may be used to find occurrences of, and apply transformations to, concise patterns which express arbitrarily sophisticated search criteria. *LOCATE* finds any pattern, while *REPLACE* first does a search, and then performs a transformation ranging from a straightforward text substitution to a complex manipulation of the located pattern. Patterns can be devised to match simple sequences such as substrings and syntactic elements, or compound constructs such as assignment applied to names whose first letter is underscored but whose subsequent letters are not.

### 7.4 Utility Functions

LOGOS is an excellent framework for storing programming utilities to aid in system development and management. Some collections of utilities are provided as part of the LOGOS system. These include:

- LOGOS 'primitives'—functions which can be utilized from within an application to programmatically request LOGOS services;
- Functions to aid in APL file management, particularly as it pertains to generating files; and
- Shell utilities, which manage the creation and use of isolated environments that help to minify interactions with the global environment.

## 8. DISTRIBUTION OF SOFTWARE

Software distribution is a critical part of the life cycle, and covers two important problems. First, it involves the release of new code to the system on which the development occurred. Second, it may additionally entail deployment of software to other systems for remote installation and integration. Because of the way in which LOGOS supports overlaid working directories and page files, the generation of both test- and production-mode systems on any one host is straightforward, and the transition from one mode to the other is mitigated. However, the problem of porting software is potentially complex. As the number of sites running APL increases, so does the need to have a simple, effective method of distributing APL software.

As a common transfer medium between systems, LOGOS uses an *export file*. Export files are ordinary LOGOS files, but have the additional attribute of being able to be moved to other machines and connected to other LOGOS file systems. An export file can be dumped to tape, transferred, and then retrieved at the receiving site using the standard SHARP APL file utilities.

Export files are created and built using the *EXPORT* command, which copies a list of pathnames to a specified export file. In order to preserve the original names, *EXPORT* transforms them by prepending the name of the export file to the full, original pathname. For example, if you export *.LOGOS.UTILS* to the file *.IPSA.UPDATE*, the pathname in the export file would become *.IPSA.UPDATE.LOGOS.UTILS*. The command to do this is straightforward:

```
◇ EXPORT .LOGOS.UTILS .IPSA.UPDATE
```

After the export file has been retrieved at the receiving site, the *IMPORT* command is used to attach it to the LOGOS file system:

```
◇ IMPORT .IPSA.UPDATE
```

Once attached, the file becomes a completely ordinary LOGOS file which may participate in standard data base operations, including environment generation. It is important to note that the *IMPORT* command doesn't actually *move* any data out of the file being imported. To do this, you can use the *MOVE* or *COPY* command to distribute the file's contents elsewhere in the file system. Often this is done only after the new software has been vetted by generating and experimenting with test versions of the affected applications.

By using the *PROCESS* command in conjunction with scripts, you can automate the building and use of export files. The following example shows the contents of a script designed to import and regenerate the *.LOGOS.UTILS* system:

```
▽ START
[1] )IMPORT .IPSA.UPDATE A IMPORT TRANSFERRED FILE
[2] )MOVE .IPSA.UPDATE . A REDISTRIBUTE NEW DIRECTORY
[3] )PROCESS .LOGOS.UTILS.INSTALL.* A REGENERATE EVERYTHING
▽
```

In addition to the use of export files for software deployment, the facility can also be used to archive applications kept in LOGOS. An export file can be built, dumped to tape, and then deleted from the LOGOS file system. Then, if the need arises, this tape can be retrieved and imported back into the active LOGOS system. Archiving provides two extremely important facilities:

- It allows customers or sites to move inactive applications from expensive, online storage, onto cheaper, offline storage.
- It allows important applications to be backed-up independently and stored in a secure location, as a precautionary measure. Then, if anything happened to the active application, all or part of the archived version could easily be retrieved for use.



## 9. MANAGING SOFTWARE

LOGOS has many facilities which offer both administrators and developers direct support in the management of software. A number of these, such as version tracking and backup, user groups, and access controls, have previously been mentioned. Several other salient features which assist in the management of software are presented below. Taken together, these features represent powerful capabilities which improve programmer consistency, working style, and productivity.

### 9.1 Registration

To meet the dynamic requirements of programmers during the intensive development and debugging phases of a project, a software library management facility is essential to provide protection against simultaneous modification of objects. This is particularly important in projects employing a large number of people, or in those whose contributors are geographically dispersed, for in both cases ensuring perfect communication among all individuals at all times is difficult and resource-consuming.

In LOGOS, a facility called *registration* lets you indicate your intent to alter an object before you have actually begun to do so. A simple command such as:

◇ *REGISTER OUT REPORTGEN*

does this for the object named *REPORTGEN*. While *REPORTGEN* is registered as being out, other users are prevented from altering it and are notified that you have the object registered. (This operational 'block' can be overridden if necessary, with automatic notification of the registrant that this was done.) When you have completed your changes, you can remove the registration qualification by entering:

◇ *REGISTER IN REPORTGEN*

Alternatively, a modifier to the *SAVE* command lets you remove registration concurrently with creating a new version of an object. It is worth emphasizing that the registration facility is designed right into LOGOS, and is not a bolt-on feature such as that commonly found in programming environments (UNIX's Source Code Control System [Roch75], for example). By being firmly integrated, registration is simple, efficient, and automatic; once an object is registered, it is so designated for everyone, not just those choosing to access it by a particular means.

## 9.2 Auditing and Tracking

LOGOS provides automatic auditing which records, for each workspace, data file, or page file generated, precisely which objects and corresponding versions comprised the generated document. This auditing data provides two facilities which are important from both an administrative and a developer's viewpoint. First, the command *USED* lets you interrogate who has referenced a particular object, either through the generation of an end environment or through a link in the file system. For example, the inclusion of the object *.DICK.UTIL.VTOM* in a workspace, or a link established to this object, both constitute references. From this perspective, the reference list for an object helps the developer assess the impact of a change *a priori*, and permits notification of those users whose systems might be affected. This markedly increases the integrity of applications across modifications which alter shared components.

The second way in which audit information is valuable is in permitting the constituents of a particular instantiation of a workspace or file to be determined. This provides the administrative capability to track down problems that might relate to the composition of a generated environment.

## 9.3 Broadcast Notes

Any LOGOS object or directory can have a broadcast note associated with it. The broadcast note is displayed each time you reference the source for an object, and each time a detailed directory listing of the object is requested. The note might typically specify words of caution, recent changes, semantic dependencies upon other objects, or even a disclaimer—anything for which automatic (and often ephemeral) notification of the user from the developer is desired. For example, if a particular function is known to possess a temporary bug or side-effect, the broadcast note is a convenient place to document the short-coming.

## 9.4 Macros and User-Defined Commands

Experienced users tend to find that there are certain command idioms which they enjoy and use frequently. The LOGOS macro facility provides a way to define such idioms, and have them be indistinguishable from ordinary commands in terms of entry syntax. A useful macro might be one which displays and cross-references a function, by calling the *DISPLAY* and *XREF* commands in turn; or one which changes named objects from test to production status by moving them to production directories and regenerating all systems which reference them.

Users may also define their own commands, which are simply APL programs that are called in a special way by the LOGOS command scanner. Being programs, user-defined commands are more general than macros and can perform arbitrarily complex tasks. They, too, are syntactically and semantically indistinguishable from native LOGOS commands. Examples of candidates for user-defined commands include a new tool to analyse LOGOS objects, and a directory list program which prints its output in a format different from *LIST*.

Both macros and user-defined commands are ordinary objects in the file system, and hence may be shared with other users. In addition to permitting customization to personal taste, these facilities can be used as management tools to encourage certain practices among users working on a common project.

## **10. LOGOS AND THE SOFTWARE LIFE CYCLE**

It has already been mentioned that LOGOS is comprised of a comprehensive set of integrated software engineering tools which are relevant to each phase of the life cycle. This integrated approach yields a synergism not found in systems which simply provide a 'grab-bag' of discrete tools. This section is a treatise on how LOGOS facilities can be applied to the diverse requirements demanded by the phases of the life cycle.

### **10.1 Specification**

The output of the first part of the development process is a functional specification document. This document is a description of what the system in question is intended to do. Specific information pertaining to *how* these goals are to be met is not included.

The functional specification frequently consists of multiple documents. The hierarchical structure of the data base can be used to impose a modular structure on these documents, based on the organization of the application system itself. In addition, use of the file system makes it possible to extract, merge, and share documents. Some of these extracts may be carried forward into later phases.

Text can be prepared and edited using the LOGOS object editor, or using any one of a number of other editors which have interfaces to LOGOS.

### **10.2 Design**

The design phase involves the production of a design specification document. This document describes exactly *how* the system will operate, by developing the relevant algorithms and underlying data structures in detail.

As with the specification phase, the mechanical preparation of this document is facilitated by LOGOS. The ability to carry forward sections of the document can be used very effectively. For instance, an extract from the design document can be inserted into the comments of a function written during the ensuing coding phase.

More significantly, LOGOS simplifies the design process itself by providing several well-defined application environments. Facilities for defining, generating, and maintaining these environments are integral to the system. For example, for large applications where finite workspace size may be a serious constraint, paging environments can be employed. Where program interaction with the global context is an issue, shells can be used. Having these facilities readily available to all applications means more consistent design among applications, improved reliability resulting from the use of tested mechanisms, and a shifted design emphasis away from these details and toward more productive issues.

### 10.3 Coding

During this period of the cycle, design is transformed from abstract descriptions into executable programs. Here, as with earlier phases, the LOGOS editor and file system are invaluable in the preparation of program source and documents. The ability for the editor to manipulate any APL object makes it possible to build tables and arbitrary data structures in a convenient fashion.

In the initial stages of prototyping, isolated system components can be replaced with dummy modules, or *stubs* [Your79]. Stubs are simple functions which usually return invariant results for the purpose of testing interdependent modules. When the implementation of a component has been completed, the stub can be replaced by its functional counterpart. The concept of stubs as a methodology for engineering the development of software works well in LOGOS, due to the system's ability to support parallel directories of functional and dummy modules.

Because the file system encourages controlled sharing, a new application can often be built in part using pieces acquired from elsewhere in LOGOS. Existing objects are easily incorporated into new applications, either through direct references or indirect links. In neither case is the object itself moved or duplicated within the file system. As a result, the benefits of updates to the source copy of an object can spread to all applications employing the utility whenever the applications are regenerated.

LOGOS is especially effective in dealing with the implementation of ambitiously large systems. Facilities such as reference lists for objects, registration, broadcast notes, directory overlays, and integrated tools all contribute in this regard.

## 10.4 Testing and Debugging

LOGOS makes many provisions for increasing the effectiveness of this stage of the development cycle. By using parallel directories to store prototype, test, and production versions of objects, a programmer can structure the application in an efficacious manner. If the prototype directory contains stubs, then, during debugging, suspect modules can easily be replaced by their stub counterparts until the offending module is located.

The testing and debugging phase is iterative, and usually involves extensive use of an editor. Being integrated, the LOGOS editor works particularly well in this capacity. Moreover, if your mode of development involves establishing a test and a production working directory, then the editor is especially propitious: When you edit an object, the test version is fetched if there is one; otherwise, LOGOS searches for the production version. But after editing, by default, the object is saved in the first (test) working directory—thereby automatically creating a test version if one didn't previously exist.

Certain of the compilation directives were designed to complement testing and debugging endeavours. For example, using special APL comments placed within a program's source, you can specify code fragments which are to be optionally included in or excluded from the program when it is generated under certain pre-specified conditions. Examples include statements which set stop or trace vectors, print diagnostic messages, sample accounting information, monitor usage, and so on. This feature makes it possible to have one version of an object which functions in both test and production capacities; the statements that will actually form a part of the function in a generated environment are dynamically determined when it is compiled.

## 10.5 Integration and Distribution

This period of the development cycle concerns itself with the integration of completed modules into a cohesive package, for release on the development system or for distribution to other hosts. In building workspaces and files, LOGOS derives much of its flexibility from the fact that APL is the generation language. Any condition or calculation expressible as an APL statement can be included in a generation script. In particular, information such as the name of a target generation file can be treated as a parameter. Proper construction of generation scripts with targets as parameters is often sufficient to permit the effect of generation to be reversed with no side-effects. Taken together with the ability to backup any file or workspace prior to regenerating it, releasing software becomes a much less critical and painful operation than it might otherwise be.

The *EXPORT* and *IMPORT* commands provide a convenient way to prepare data for movement off one system, and to establish it within the LOGOS hierarchy on another. Software distribution to other hosts is therefore nearly as straightforward as distribution to the native host.

## 10.6 Maintenance

Maintenance involves all activities related to the continued operation of a software package. These include the correction of problems as they occur, as well as the implementation of new capabilities. Adding these capabilities requires returning to earlier stages and going through the entire process again, thus the term 'life cycle.'

This final stage is crucial to the success of a software project. It has been estimated that more time and resources are spent here than in any of the other phases. Many of the features described in earlier sections have a direct bearing on the ease of maintaining applications stored in LOGOS. Unilateral access to the specification documents, design documents, and multiple versions of programs is an indispensable feature.

Of particular utility during the resolution of problems are the various items of information maintained in the repository for an object. For example, the change journal is a document which, by convention, contains historical information pertinent to the object; backup copies of earlier versions of objects assist in the tracking of components in distributed software packages; and reference lists remove the dangerous guesswork of trying to deduce who might be affected by a planned change.

## 11. EXTENSIBILITY OF DESIGN

One of the main contributions of a programming environment is to simplify the task of writing more sophisticated software applications. With LOGOS, this was achieved in part by adopting the view that the programming environment is a *base* on which other facets of software development can be built. This view can be justified in several different ways:

- From the specification perspective, it forced rationalization, simplification, and consistency constraints upon design, yielding a better product. A system should be judged nearly as much by what isn't included in it, as by what is.
- The adaptability of a system is important, for there is no panacea for all programming ills. Simply put, nothing can please everyone. LOGOS therefore makes it possible to customize many aspects of its practice: profiles, command and modifier abbreviations, macros, and user-defined commands are examples of customizations which can be used to advantage. With the latter two, innovative users can write their own commands, or alternate versions of existing commands, perhaps improving upon them.
- Commands, tools, or general programs written by one person are often shared with others and act as building blocks for their software. The uniformity of the file system and its powerful access controls encourage this methodology.
- Without the provision for extensibility, a product as sophisticated as a programming environment is likely to atrophy quickly as the needs of its user base outstrip its capabilities. To this extent, we hope and anticipate that LOGOS will continue to evolve with changing software engineering and user practices.

Too much flexibility can overpower novices. However, in LOGOS, the details of this flexibility are ensconced by simple external interfaces, and users who are happy with the system and its defaults as provided need not concern themselves with customization at all.

## 12. CONCLUSIONS

Software development in the APL environment is a complex task which is exacerbated by the existence of varying object data types and multiple storage media for these objects. The design of LOGOS espouses the view that a comprehensive programming environment to facilitate the development, debugging, maintenance, and distribution of APL applications can be constructed through the use of a consistent external interface and tightly integrated software which is intimately familiar with APL intricacies. The homogeneous user interface effectively eliminates firewalls normally encountered between independent collections of tools, and makes the transition from one phase of the software life cycle to another a subconscious effort.

From one perspective, LOGOS may be sensibly viewed as an operating system. We don't necessarily like to think of operating systems in the same breath as the users of an APL system (for the APL system does an excellent job of shielding us from the vagaries of the true machine interface), but nevertheless the analogy is worthwhile. As any operating system should, LOGOS provides a file system for managing information, and facilities for writing and running commands and programs. There are protection mechanisms, system generation facilities, resource management aids, and automatic recovery features for exogenous events. All of this is available through both an interactive and a non-interactive access method, the latter being achieved through primitive, self-contained LOGOS utilities. When it is all tied together, LOGOS represents a considerable amount of software designed to provide a wide range of services geared towards simplifying the task of producing systems in APL.

The true test of success of a programming environment—or indeed of any piece of software—is the level of usage it receives, and in this regard the most important perspective from which to view LOGOS is the user's. LOGOS is easy to use, and easy to begin using. It can be integrated slowly into the work patterns of individuals, and it can help with the maintenance of existing systems immediately. We hope that people who are involved with APL software will, of their own volition, come to use LOGOS for its structure, its facilities, its adaptability, and, most significantly, for its ability to solve their problems.

## REFERENCES

- Deli84      Delisle, Norman M., et al. "Viewing a Programming Environment as a Single Tool." *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Development Environments, Software Engineering Notes*, 9:3 (May 1984), pp. 49-56.
- Demp81      Dempsey, Mark R. and L.H. Goldsmith. "A Regular Expression Pattern Matching Processor for APL." *APL81 Conference Proceedings and APL Quote Quad*, 12:1 (September 1981), pp. 94-100.
- DGC82      *AOS Command Line Interpreter*. Publication 093-000122-05, revised, Data General Corporation, Westboro, Mass., May 1982.
- Gold81      Goldsmith, Leslie H. *Dynamic Protection of Objects in a Computer Utility*. M.A.Sc. Thesis and Technical Report CSRG-130, Computer Systems Research Group, University of Toronto, Toronto, Canada, April 1981.
- Goldb84      Goldberg, Adele. *Smalltalk-80: The Interactive Programming Environment*. Addison-Wesley Publishing Co., Reading, Mass., 1984.
- Golds84      Goldschlager, Leslie M. *A Computational Theory of Higher Brain Function*. Department of Computer Science, Stanford University, Stanford, Ca., April 1984.
- Howd82      Howden, William E. "Contemporary Software Development Environments." *Communications of the ACM*, 25:5 (May 1982), 318-329.
- IPSA83      *APE: APL Program Editor*. I.P. Sharp Associates Limited, Toronto, Canada, May 1983.
- IPSA84      *WSDOC:SHARP APL Workspace Documentation Facility Users' Guide*. I.P. Sharp Associates Limited, Toronto, Canada, August 1984.
- Jens79      Jensen, Randall W. and C.C. Tonies. *Software Engineering*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1979.
- Kern81      Kernighan, Brian W. and J.R. Mashey. "The UNIX Programming Environment." *Computer*, 14:4 (April 1981), 12-24.
- Medi81      Medina-Mora, Raul and P.H. Feiler. "An Incremental Programming Environment." *IEEE Transactions on Software Engineering*, SE-7:5 (September 1981), 472-482.
- Oste81      Osterweil, Leon. "Software Environment Research: Directions for the Next Five Years." *Computer*, 14:4 (April 1981), 35-43.



- PEG81      *A Quick Reference Guide to the SHARP APL Implementation of the APL Workshop*. The Planning Economics Group, Boston, Mass., November 1981.
- Ritc78      Ritchie, Dennis M. and K. Thompson. "The UNIX Time-Sharing System." *The Bell System Technical Journal*, **57:6** (July-August 1978), part 2, 1905-1929.
- Roch75      Rochkind, M.J. "The Source Code Control System." *IEEE Transactions on Software Engineering*, **SE-1:4** (December 1975), 364-370.
- Teit81      Teitelman, Warren and L. Masinter. "The Interlisp Programming Environment." *Computer*, **14:4** (April 1981), 25-33.
- Your79      Yourdon, Edward and L.L. Constantine. *Structured Design*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1979.
- Wass81      Wasserman, Anthony I. "Automated Development Environments." *Computer*, **14:4** (April 1981), 7-10.



# **The SHARP APL Environment**

Richard Lathwell, Manager, Research and Development, I. P. Sharp Associates

## **Abstract**

An APL environment as a means for controlling computing processes was formulated when a research group at the IBM Thomas J. Watson Research Center developed an experimental computer system, one aspect of which was the ability to achieve independence of hardware and software architectures. Later, the environment was extended so that system dependence could be obtained when it was desirable. The SHARP APL environment is an embodiment of those ideas; one which continues to evolve and develop with changing technology.



In 1966, a group of people conspired at the IBM Thomas J. Watson Research Center to produce an interactive timesharing system called APL\360. APL\360 was implemented on a System/360 computer and consisted of an interpreter of the APL language and a time-sharing supervisor. This implementation was unique in many ways, but perhaps the most important difference resulted from the desire of the designers to view the underlying computer as a mechanism for providing as nearly as possible an abstract environment. The precedent for this radical approach was probably set earlier by Adin Falkoff and Ken Iverson who refused to be bound by the limitations of typewriters and therefore designed a type element for IBM SELECTRIC typewriters. The alphabet was limited to a single case and the graphic positions thus made available were used for language symbols which were chosen with an eye to how they overstruck one another to compose additional characters.

APL\360 introduced 'the APL environment': an implementation of an abstract APL machine which could be used to describe and control a computing process (write and run a program in the vernacular). The system was used from typewriter-like terminals, and users did not need (and were not able) to interact directly with any other software. Users had the illusion of working on their own dedicated machines and were typically quite unaware of other activity on the same computer, a point emphasized whenever the system crashed when sixty people would call simultaneously to explain how they each crashed it.

One of the more important aspects of the APL environment is that it recognizes the need for portability both of programs and of skills, and one of the reasons for the easy acceptance of APL systems is most certainly the ease with which elementary arithmetic and algebraic skills learned in school can be applied. I've occasionally wondered what it was that the original project members had in common since they are each quite different from the others, and I've only been able to think of one thing—we all used the language before it was implemented and still do. (When I asked this question of Ken Iverson, his opinion was that each of us was both stubborn and eloquent.)

The original APL\360 implementation depended solely on System/360 hardware and was unfettered by any operating system or, more properly, APL\360 contained its own operating system which supported nothing except the APL interpreter, workspace libraries, and terminals. While independence of the environment was an advantage in a great many application areas, there were two areas of particular interest from which it was excluded: it was impossible to directly interact with a System/360 program, and it was impossible to use the computer for other things with APL running. The use of APL was increasing rapidly, and interest in these areas increased accordingly because of the desire to be able to analyse and repair the software while APL was running. There was also a desire among the system programmers to be able to work during daylight hours. In order to address these areas, two steps were taken—one seeming tiny and trivial at the time but of major consequence, and the other inevitable.

The step which seemed of little consequence and merely a system programming convenience was the introduction of a primitive function, the I-beam, to permit a crude interaction

between an APL program and the underlying software. Monadically, it returned integer vectors representing performance histograms maintained by the APL\360 supervisor, and dyadically it permitted the computer storage to be displayed and altered. By this simple means it became possible for an APL program to examine and control its environment. Thus, the notion of a system function was invented more or less by accident. Through the I-beam function, it became possible to extend the APL environment to include the storage of its host computer and hence it became possible to write APL programs which were explicitly machine dependent.

The second inevitable step was to modify APL\360 so that it could operate in conjunction with the IBM Disk Operating System, permitting the computer to be shared with other programs. An implementation in the mainstream IBM Operating System soon followed. These two implementations provided an architectural approach which proved quite successful in the following years—an APL environment operating as a subsystem withing a larger host operating system environment.

This, then, was the stage that development had reached when APL\360 was made available as an IBM product in 1968. It's interesting to note that APL\360 DOS operated quite satisfactorily on a System/360 Model 50 with 256K of storage—a computer small in comparison with the current single-user IBM PC XT/370. The system provided a respectable interactive APL environment. Except for the I-beams, which were restricted to privileged users, and a few functions which allowed the control of workspace parameters such as the index origin, applications were limited to the workspace and to the information that could be communicated to or from a terminal. In order to be commercially viable, the language and its implementations required the ability to move large quantities of information between the workspace and the environment.

At this point, a dichotomy appeared in the development of APL environments. The needs of the language could be viewed in two ways:

- storage needs—the need to store large collections of data represented as APL objects
- communication needs—the need to exchange information with programs which might be outside of the APL environment.

Organisations such as I. P. Sharp Associates chose the former view and extended the language by enlarging the APL environment, that is, by adding system functions to the language to increase its domain. The most obvious of these extensions are native file systems which are essentially storage architectures, although communication between programs can be achieved by means of shared files. Communication with non-APL environments is achieved by utility programs to import and export information to and from the file system, that is, by moving information to and from the APL environment. System functions were also added to provide a variety of other useful facilities such as report formatting similar

to that provided by other contemporary languages, the ability to initiate new APL tasks, and the ability to intercept and programatically recover from execution errors and other events that might otherwise require intervention by a user.

IBM, on the other hand, chose to pursue the problem of communication with the APL environment and developed shared variables on the assumption that access to large volumes of data could be achieved if an APL program had the ability to communicate with a program which managed files. Rather than extending the APL environment to encompass more diverse objects, this approach provided doorways. Shortly after shared variables were implemented, the notion of auxiliary processors—programs written in other languages capable of communicating with APL programs—was conceived to permit the construction of ‘hybrid’ systems of different environments interconnected by communication channels.

The advantage of the former approach is that it exploits machine independence by extending the APL environment. Its disadvantage, however, is that it requires strict care in designing the language extensions so that the number of functions remains reasonable. It also places a heavy burden on APL system development programmers who must transport the complex environment as host environments evolve. The advantage of the latter approach is that it allows explicit host dependence when such is desired; its disadvantage is that host-dependent programs often cannot be moved easily from one environment to another.

Until the mid 1970s, SHARP APL was developed primarily for I. P. Sharp’s timesharing service. For a variety of reasons it became desirable to distribute the SHARP APL System to other sites and this resulted in quite strong market pressure for compatibility with IBM APL products. At the same time, various national standard-writing organization began serious efforts to produce an international standard for APL implementations. As a result of both of these forces I. P. Sharp undertook the development of shared variables within the SHARP APL environment. The result is the current SHARP APL environment—an amalgamation of the environment extended through system functions, and doorways to other environments through shared variables. Inevitably, the two kinds of facility have been used in various combinations to produce some extremely sophisticated applications. For example, the ability to be able to start additional tasks which can share variables with the initiating tasks provides all APL users with the ability to construct or use APL multiprocessing systems. A host of utility service tasks which are written in APL and run continuously has greatly enhanced the services available to APL programs.

The SHARP APL environment is most strongly influenced by very large IBM computers operating with very large IBM operating systems, but in 1983 a research project was undertaken to evaluate a single-user SHARP APL environment on an IBM PC running PC DOS. This proved to be a viable strategy, and with the advent of the IBM PC XT/370, performance similar to last year’s small mainframe computers was realized.

The SHARP APL environment is a concept—a strategy for organizing the resources of a computer for its users which continues to evolve with the underlying technology, and as new application areas are entered.





# **Exploiting Networks**

Joey Tuttle, Vice-President Development, I. P. Sharp Associates

## **Abstract**

All useful applications of computers involve the concept of a network that is a "group or system of electric components and connecting circuitry designed to function in a specific manner." Applications may range from high-speed interactive computing to movement of bulk data between systems for distributed data bases or for printing. This session will describe the various levels of networks available to the user of SHARP APL. Several important concepts will be covered: shared-variable interface between SHARP APL and IBM SNA/SDLC facilities; IBM network protocols for supporting user workstations or establishing contact with other resources in the IBM system environment; local area networks for fast, economical movement of data between workstations; and exploiting IPSAN-ET resources for worldwide communications. Case studies of practical applications will be presented to illustrate how to plan and implement applications that involve network communications.



The idea of a network is central to any use of computing facilities. The manifestations range from a connection that reaches half way around the world to the wires connecting the keyboard of a personal computer to its processing unit. In many cases there are several choices in the method used for communication in a network. The rules used for a particular kind of network communication are called the network protocol. A single network may use many different protocols at various points along the communication path. Devices communicating with a network may use a variety of protocols and have very different speeds of communication.

An effective electronic network is essential in the implementation of the Global Information Centre concept. In this session of the 1984 APL Users Meeting the speakers explore the changing technology of networks. Mr. Bob Bethenod of IBM Corporation will discuss ideas that IBM is considering for communication products in the future. Dr. Harry Saal of Nestar Corporation will present ideas about Local Area Networks and the facilities and capabilities they make available to business users via locally connected personal computers. David Chivers of I. P. Sharp Associates will describe the communication network products available from I. P. Sharp Associates.

A common objective of all the points of view represented in this session on Exploiting Networks is to allow the users of quite different equipment to use data of common interest. One of the most important aspects of networks is to match the needs of various terminals or workstations to the network and hence to resources on the network. In the Nestar Local Area Network, seven distinct protocol layers are combined together to insure that changes in communications technology cause minimum disruption to application systems depending on the network. It is interesting to consider how the current collection of communication equipment and techniques came into existence.

## **WHERE NETWORKS CAME FROM**

Probably the most widespread idea of network communication involves the use of a terminal connected to a processing unit some distance from the terminal. Early examples of such terminals include the TELEX machine. These teletype machines used a communication encoding technique using 5 bits per character known as Baudot Code. Typical Baudot Code transmissions allow 52 different characters to be transmitted at about 10 characters per second. As users insisted on having more than the limited single case alphabet available on a teletype, an extension called American Standard Code for Information Interchange (ASCII) came into wide use. ASCII uses 7 bits to represent 128 different possible characters (actually, 96 characters with the rest of the codes used for control purposes).

Other early terminal devices include the IBM terminals based on the Selectric typewriter mechanism, such as the IBM 1050 and the 2741. The 2741 in particular was aimed at bringing the power of a central computer into the office environment. This was a departure from the idea of point-to-point communication provided by TELEX machines. The 1050 and 2741 operate at 134.5 baud and use 6 bits to represent 64 unique codes. To extend the number of characters available, the IBM protocol uses 2 of the codes to represent up-shift and down-shift (similar to the Baudot Code) thus making it possible to represent the 88 or so characters that appeared on the Selectric type element. The IBM 1050 and 2741 achieved a data rate of about 14.8 characters per second.

It is interesting to note that the binary representation of characters on the 2741 is quite different from a teletype, and beyond that there were two distinctly different representations depending on whether a particular 2741 was a 'BCD' machine (matching the characteristics of the 1050) or a 'correspondence' machine that was similar to the codes used in office typewriter products. Many of the early IBM timesharing systems supported one or the other but not both types of IBM terminal!

All of the terminals mentioned so far use an asynchronous protocol for communicating. The term asynchronous arises because such terminals send one character at a time to a remote receiver. There is no regular speed with which characters are sent but rather the interval between arrival of characters typically depends on the speed of a typist. Because of the start and stop nature of asynchronous communications the receiver must determine for each character what was being transmitted by inspecting the few bits representing a single character. Noise on telephone lines may introduce spurious bits into characters being transmitted asynchronously and frequently such spurious data is incorrectly interpreted by the receiver and results in (perhaps undetected) errors in the data.

There are various techniques to make users aware of transmission errors. One such technique is the use of full duplex communication protocol and 'echoplex' for asynchronous terminals. Echoplex means that characters typed by the user are sent to the remote computer (but not to the local terminal display) and it sends back characters that it receives to be displayed on the terminal. If the characters that are displayed on the terminal match the ones typed in, it is very likely that no errors occurred during the transmission. Some systems also take advantage of the full duplex protocol to allow the remote computer to invoke terminal control functions (for example graphics or screen management) as a result of some arbitrary keystroke on the user terminal. This technique is very effective when the terminal is relatively tightly coupled to the processing unit via a high speed link.

The communication between the keyboard and display of a personal computer is very much like a locally attached full duplex terminal since typed keystrokes go to the processing unit and it (perhaps) places them on the display screen. Having the ability to process each user keystroke in an arbitrary way is one of the biggest advantages of a personal computer. This immediate processing of input is what makes systems like spreadsheets, interactive graphics, and games so attractive on personal computers. Of course, disadvantages of personal computers include their limited ability to store very large data files and do large amounts of highspeed processing. Access to remote data bases and processing power is important to business users of personal computers.

It is interesting to consider what a host supported echoplex application program would look like when connected to a remote system that involved a couple of earth satellite links (e.g., perhaps between Hong Kong and Europe). The physical speed of light would impose a delay of more than a second to display typed characters. This kind of delay makes it important to use local processing power to insure accurate communication.

A rather different style of communication is represented by the IBM 3270 family of terminals. In 3270 terminal systems, several user terminals are typically connected to a terminal controller which is in turn connected to some mainframe computer. Each of the user display stations is connected to the controller by a coaxial cable and communicates with it at a rate of several million bits per second. The controller may be directly attached to a mainframe computer and communicate with it at several hundred thousand characters per second. Alternatively the controller may communicate via a network where the rate is somewhat lower—typically 400-4,000 characters per second.

The 3270 terminal controller is actually a powerful processor specially programmed to do certain kinds of terminal control. Application programs which use 3270 devices must describe the format of the display screen to the controller. The typist at a 3270 typically types input into one or several input fields and this activity is completely local to the terminal and its controller. When the 3270 user strikes the enter key (or one of the program function or attention keys) the controller sends information to the remote system. This batch transfer of information is quite different from the character-at-a-time mode of asynchronous terminals.

Because of the batch nature of data from 3270 controllers, a synchronous transmission protocol is used when they are network connected. Such synchronous protocols send data at an exact fixed rate and have built-in error checking and recovery mechanisms. Even if the controller is connected to a remote system via relatively noisy telephone lines, data transmission is typically error free (perhaps at the cost of some slow down of transmission resulting from resending the data).

## **WIDE AREA NETWORKS**

In the mid 1970s, I. P. Sharp Associates developed a network to allow users around the world to access remote computers (both I. P. Sharp timesharing services and inhouse installations of I. P. Sharp systems). The IPSANET network is a packet-switched network. Packet-switching combines the ideas of previously mentioned terminal communications systems. Typically, asynchronous terminals are connected to a node (processor) on the network via a dial-up dataphone connection. As the user types on the terminal, the local node accumulates the input characters. When about 30 characters are accumulated, the local node puts them into a packet and dispatches the packet through the network to its destination system. Of course, if the user indicates end of input before accumulating 30 characters

(e.g. 2+2), then the data is immediately forwarded to the host system. If the terminal user desires, echoplex may be used to validate transmitted data—but note that the characters on the terminal are sent back from the most local processor in the network, not the host mainframe computer.

Since echoplex can be used and the network node can be as close to the end user as required for a particular application, the difficulties due to noise introduced errors are greatly reduced. Data transmission between nodes in IPSANET uses synchronous protocols and error-free transmission to the host is assured. Since full duplex terminal operation is only with the local node, it is impossible for the application program to take advantage of key-at-a-time input. On the other hand, the processing power of the local node is used to support a diversity of terminal types and speeds. The terminal types supported range from TELEX machines to 134.5 baud IBM terminals, 300 to 9600 baud ASCII terminals, and synchronous terminals sometimes known as Remote Job Entry (RJE) systems. RJE terminals are used for bulk data transfer such as reading and creating magnetic tapes remotely and supporting relatively fast printer systems such as the IBM 6670 Information Distributor (a laser printer).

It is clear that the most useful and flexible protocol for wide area networks is currently the packet-switching technique. The public service networks (e.g. TELENET, TYMNET, and DATAPAC in North America) use packet-switching techniques. It is very desirable to be able to move data from one network to another and there are internationally standard protocols for doing that (one is called X.25). The public service nets are interconnected to IPSANET via X.25 interfaces allowing a terminal user with access to a public network service to go through one of these 'gateways' to IPSANET and from there on to use applications in a SHARP APL system.

With all of the possible ways to interconnect a given terminal to a given system, it is important to keep the principal objective of a good network foremost in mind; that is, the best network is one of which the user is unaware.

## **NETWORK TO SOFTWARE LINKS**

A network should provide the simplest possible way of connecting the users terminal, word processor, or personal computer to the desired system and then the network should simply handle the problems of error free communication without surprising the user. This transparency is a principal design goal of IPSANET. The issues of effective communication extend beyond the job of connecting a keyboard to an application program and it is important for the programming environment used for developing and running applications to have effective tools for communication. Generalized communications was the basis for the development of shared variables in APL. Using the simple but powerful mechanisms provided by shared variables, many different kinds of network communication systems can be implemented.

SHARP APL has the ability to support users of IBM 3270 terminal devices in addition to the originally supported asynchronous terminals. This ability was added by devising a 'session manager' for 3270 devices using the IBM System Network Architecture and Virtual Terminal Access Methods (SNA/VTAM). The session manager program appears as a standard application program to the VTAM network. The current 3270 session manager is called the Intelligent Display Station Handler (IDSH) and has the responsibility of exploiting available functions of 3270 terminal devices and also dealing with moving data to and from the APL system. To facilitate such programatic use of APL (as opposed to the older idea of connecting a user terminal directly to an APL task) a new interface to SHARP APL was created. The data exchanged between SHARP APL and IDSH is all done via shared variables.

An Auxiliary Processor (AP) is available in SHARP APL that provides a program with direct access to the APL system. When a program accesses SHARP APL in this way, it may create an STASK which is simply an active APL session with all communications via a shared variable. In this way, it is straightforward to construct programs that greatly extend the ability of APL to support users through new styles of network connections. This is the way IDSH connects 3270 users to SHARP APL.

In addition to the 3270 SNA/VTAM session manager (IDSH) there is another AP that provides access to SHARP APL for asynchronous terminals that may be operating in the IBM network environment. The main point of this is that most of the effort put into SHARP APL and IPSANET has been to provide the essential primitive tools that enable flexible and effective communications systems to be built.

## SOFTWARE TO NETWORK LINKS

A slightly different view of networks is obtained if one looks at the idea of using a program to initiate activity on a network. To this end, SHARP APL provides an AP called APL VTAM Access Method (AVAM). AVAM permits an APL program to communicate directly with VTAM in the host machine where the program is running. Using this gateway, it is possible for an APL program to initiate and manage a work session in some entirely different system which is available as a VTAM application. This ability is a unique and very powerful way of establishing network gateways.

Recent work has been going on to allow a user on one SHARP APL system to effectively share a variable with a user on a physically separate SHARP APL system via IPSANET. This will provide a very simple way for a program in one machine to use data from another machine or send data to another machine. The extension of shared variables into the network will facilitate writing applications that provide data in the Global Information Centre environment.

## LOCAL AREA NETWORKS

Most of the preceding discussion has been about the problems of communicating in a mainframe oriented network environment. Obviously, personal computers are becoming extremely important in the data processing world. Many uses of personal computers are not very personal. That is, most of the time, work done at a workstation (personal computer) results in data of interest to other people. Sharing such data is the principal objective of Local Area Networks (LANs).

Actually, a LAN is simply the same sort of communication facility as a teleprocessing network (including the use of packet techniques) but with dedicated very highspeed communication links. The most common form of connection for LANs is a coaxial cable very similar to the one used by IBM 3270s and their controllers. The main thing that distinguishes LAN workstations from 3270 terminals is that LAN stations are general purpose processors (personal computers) themselves. However, this distinction becomes less clear when considering the IBM 3270/PC series of machines.

A LAN usually consists of several connected workstations and may additionally include a device called a file server which acts as a central resource for data storage. In addition to the idea of connecting several workstations to a common data store (either in a workstation on the LAN or a special file server device), some stations on the LAN may act as gateways into other networks. The gateway might be via a connection similar to a 3270 into SNA/VTAM or through X.25 or even a simple asynchronous terminal.

The SHARP APL system for the IBM/PC was designed with the idea of compatibility with mainframe concepts clearly in mind. The APL shared file system can easily be adapted to operate on a LAN file server. SHARP APL/PC uses the shared variable concept to communicate between processes in the PC and this facilitates adding new capabilities. The current SHARP APL/PC system provides a way to download and upload data and programs to a mainframe APL system.

It is an objective of development work at I. P. Sharp Associates to allow programs written in APL to communicate (share variables) with other programs. This communication may occur between programs in the same machine, a neighbor machine on a LAN, or two machines thousands of miles apart. The effect to the end user is that needed information is available in the Global Information Centre environment with a minimum of interference from the many different techniques and protocols used to provide the end result.





Blank lined area for writing.